Universidad Zaragoza
1542

Proyecto Fin de Carrera

Rendering Ocean Wave Simulations

Autor

Javier Delgado Aylagas

Director: Jeppe Revall Frisvad
Ponente: Dr. Diego Gutiérrez Pérez

Escuela de Ingeniería y Arquitectura
2013

Resumen

Este proyecto ha sido desarrollado en el departamento DTU Compute de la Universidad Técnica de Dinamarca durante un intercambio Erasmus.

El objetivo de este proyecto es construir un entorno de renderizado para el simulador oceánico desarrollado en el departamento DTU Compute de la Universidad Técnica de Dinamarca [EKBL09]. Dicho simulador permite generar olas oceánicas con una gran variedad de configuraciones almacenándolas en diferentes formatos de fichero.

Este proyecto utilizará dicho simulador y permitirá importar dichas simulaciones y renderizarlas proporcionando un aspecto realista al agua generada. Para ello se transforman las simulaciones al formato OBJ utilizando Matlab para que pueda ser leído por el entorno de renderizado.

Posteriormente, se generará una salida visible de las olas generadas en el simulador, y es aquí donde se implementarán y aplicarán diferentes técnicas de renderizado para obtener un aspecto lo más realista posible. Esta parte se basa en su mayor parte en raytracing [App68], aunque se combina con otras propiedades como photon mapping para mejorar el aspecto del agua. Este apartado ha sido desarrollado en su totalidad sobre C++.

Además, se incluirán como resultados diferentes simulaciones, tanto de imágenes como de vídeos, siendo una de las simulaciones proporcionada por la empresa Force Technology con sede en Kongens Lyngby (Dinamarca).

Finalmente, se analizarán las limitaciones del proyecto y se plantearán mejoras para que pueda ser continuado en el futuro.

Agradecimientos

En primer lugar, quiero agradecer a mi supervisor Jeppe Revall Frisvad por toda su ayuda a lo largo del de desarrollo de este proyecto y también por facilitarme el entorno de trabajo, que ha simplificado considerablemente la implementación de la aplicación, y también a Diego Gutiérrez por sus comentarios para poder escribir esta memoria y por hacer de ponente para este proyecto.

En segundo lugar, quiero agradecer a Allan P. Engsig-Karup por haberme facilitado el simulador y también toda su ayuda para poder utilizarlo correctamente, al igual que a Stefan Lemvig Glimberg, sin cuya ayuda no podría haber utilizado dicho simulador.

También quiero agradecer al departamento DTU Compute el permitirme haber desarrollado mi proyecto, y especialmente a J. Andreas Bærentzen junto con Jeppe Revall Frisvad por sus comentarios y ayuda proporcionada en las reuniones semanales que se realizaron durante todo el desarrollo.

Por otra parte, también quiero agradecer a Diego Gutiérrez sus comentarios para poder escribir esta memoria y por hacer de ponente para este proyecto.

También quiero agradecer a la compañía Force Technology con sede en Kongens Lyngby su interés en este proyecto y su participación proporcionando algunos de sus modelos y simulaciones.

Por último, quiero agradecer tanto a la Univesidad de Zaragoza como a la Universidad Técnica de Dinamarca el haberme permitido desarrollar este proyecto durante mi estancia Erasmus en Dinamarca.

Índice general

Índice de figuras

Capítulo 1

Introducción

El agua generada por ordenador es un elemento muy utilizado hoy en día. Tiene una gran variedad de aplicaciones, aunque su principal uso es en producciones audiovisuales como películas o anuncios siendo también utilizada en videojuegos. Sin embargo, también hay gran variedad de compañías que lo utilizan para utilidades internas o simulaciones.

Este es el caso de la empresa Force Technology ubicada en Kongens Lyngby (Dinamarca), que dispone de simuladores donde se pueden entrenar los futuros pilotos de barcos, los cuales disponen de grandes cantidades de agua generada por ordenador de cual desean mejorar su aspecto. Este proyecto, buscará en la medida de lo posible mejorar la visualización de los simuladores de la citada empresa.

El aspecto del agua puede ser muy difícil de generar, y el proceso puede dividirse en dos partes. La primera es la geometría del agua, que debe ser actualizada en cada fotograma. El segundo paso consiste en renderizar dicha geometría para dotarle de un aspecto realista. Es este segundo paso el que se aprovechará de las propiedades del agua como material para darle el acabado deseado.

En este proyecto el primer paso de los descritos anteriormente se realiza utilizando el simulador desarrollado por A. P. Engsig-Karup, Morten G. Madsen y Stefan L. Glimberg en el departamento DTU Compute de la Universidad

Técnica de Dinamarca [EKBL09] [EKMG12].

El objetivo de este proyecto es dotar de un entorno de renderizado a dicho simulador, tomando como entrada la información generada en el simulador, y teniendo como salida imágenes PNG con la simulación renderizada de forma realista. Además, este proceso se divide en otras dos partes. La primera parte consiste en generar una entrada compatible con el entorno de desarollo. Para ello, se creará una función en Matlab que genere un fichero del formato Wavefront OBJ partiendo de los ficheros binarios devueltos por el simulador, mientras que en la segunda parte se desarrollarán las técnicas de renderizado correspondientes.

Este proyecto estudiará las propiedades más importantes del agua y que van a ser utilizadas en el mismo. Algunas de ellas son el color del fondo marino y su distancia a la superficie del agua, aunque también se tendrá en cuenta el cielo y el entorno ya que serán reflejados en parte por el agua.

El motor de renderizado está basado en raytracing [App68] y se completa con photon mapping [NJC00] para poder visualizar las cáusticas en el fondo marino. Dicho motor también ha de ser ajustado para optimizar el aspecto del agua en función de la entrada. Además, permitirá generar secuencias de imágenes de la misma manera que el simulador. De esta manera, también se podrán generar vídeos utilizando software externo.

Este entorno de renderizado es el que permitirá a la empresa Force Technology, anteriormente mencionada, a mejorar sus simuladores ya que el simulador allí utilizado es el mismo que el que se ha utilizado en este proyecto.

Este proyecto utiliza el entorno de visualización utilizado en el curso Phisically Based Rendering de la Universidad Técnica de Dinamarca para implementar las diferentes técnicas.

1.1. Resultados esperados

Este proyecto tiene dos partes diferenciadas. La primera tiene que ver con el simulador. En esta parte se explica el funcionamiento del simulador. Además, en esta parte se explica como ha sido transformado el fichero devuelto por el simulador en un fichero Wavefront OBJ.

La segunda parte incluye tanto los ajustes realizados al entorno para poder tratar las simulaciones correctamente como la etapa de sombreado. Por un lado, el entorno ha sido ajustado para proporcionar resultados más precisos y, además,

añade elementos como el fondo marino en caso de que este no lo proporcione el simulador. Por otro lado, se han añadido todos los sombreadores que han sido necesarios en el proceso, comenzando desde los más básicos hasta los más complejos, siendo algunos de ellos combinados en el proceso.

Finalmente, la salida del proyecto en su conjunto, es un número variable de imágenes, las cuales pueden ser combinadas para generar secuencias de vídeo con ayuda de aplicaciones externas a este proyecto.

Un ejemplo de la salida de la aplicación se puede ver en la figura 1.1

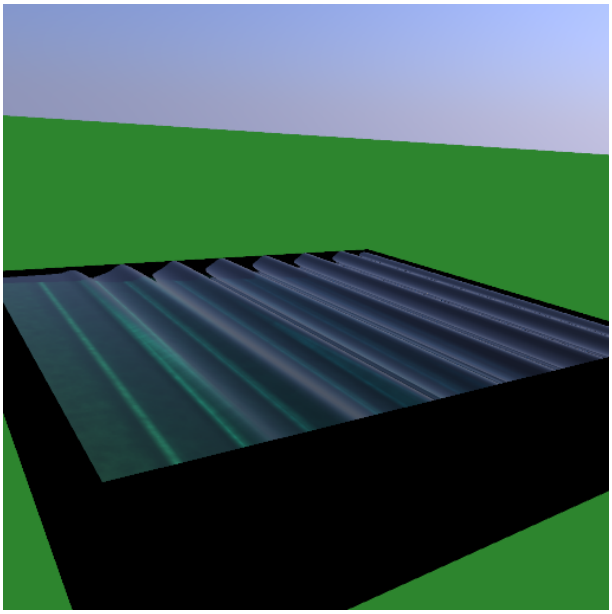

Figura 1.1: Ejemplo del resultado de la ejecución. En la imagen se puede apreciar el fondo marino, cómo la profundidad afecta al color del agua, y también la las causticas generadas por las olas.

1.2. Estructura del documento

El contenido del resto del documento está organizado como se detalla a continuación:

El proceso de renderizado. Este capítulo contiene toda la información rela-

cionada con el simulador y el entorno de desarrollo. También explica en detalle las funciones de conversión de formatos creadas en Matlab y los ajustes realizados al entorno de renderizado.

Sombreado. Este capitulo explica en detalle los diferentes sombreadores y técnicas utilizadas en cada uno de ellos.

Resultados. Esta sección detalla la información de rendimiento de diferentes simulaciones. Para ello se han utilizado dos simulaciones que darán lugar a dos posibles vídeos, y otra procedente de una simulación realizada por la empresa Force Technology y que será renderizada en este capítulo.

Conclusiones. Finalmente, en este capítulo se explican las conclusiones y posibles desarrollos futuros de la aplicación.

Capítulo 2

El proceso de renderizado

El agua es un elemento habitualmente crítico allá donde se utiliza, ya sea en videojuegos o películas, pero su nivel de detalle mejora enormemente el realismo de una escena. Además, es un problema computacionalmente complejo, y por tanto, es muy difícil de obtener en tiempo real [JL04] [Kry05]. Por esa razón, este proyecto se va a centrar en obtener una apariencia del agua realista dejando el hacerlo en tiempo real para futuros desarrollos.

Este capítulo explica como ha sido organizado el trabajo en el proyecto. En primer lugar, el simulador genera matrices de puntos exportadas como ficheros binarios. Después, estos ficheros se han de transformar al formato Wavefront OBJ. Este paso se realiza en Matlab. Finalmente, los ficheros OBJ se importan en el entorno de renderizado, el cual, utilizado según se detalla en el siguiente capítulo, generará los ficheros de imagen.

El proceso completo que se cubre en este proyecto se puede ver en la figura 2.1.

En este capítulo se va obviar el proceso de sombreado, ya que, debido a su extensión, será explicado en un capítulo específico.

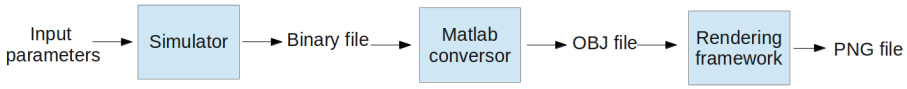

Figura 2.1: Proceso cubierto por este proyecto

2.1. El simulador

Esta sección pretende situar al lector en el contexto del proyecto, cuyo objetivo es crear un entorno de renderizado para el simulador desarrollado por A. P. Engsig-Karup, Morten G. Madsen y Stefan L. Glimberg en el departamento DTU Compute de la Universidad Técnica de Dinamarca [EKBL09] [EKMG12].

El simulador utilizado ha sido desarrollado en FORTRAN y se utiliza sobre máquinas UNIX. Consta de dos versiones, una para CPU y otra para GPU. En este proyecto la versión utilizada ha sido la de CPU.

Este simulador es el mismo que utiliza la empresa Force Technology en sus instalaciones, de manera que sus modelos serán compatibles y el entorno podría ser utilizado en sus instalaciones. De esta manera, en el capítulo 4 se ha utilizado una de las simulaciones generadas por Force Technology utilizando dicho simulador junto con uno de los modelos de barcos de los que disponen.

Al utilizar el simulador, hay una gran cantidad de parámetros que se proporcionan en un fichero de entrada y que permiten generar una gran variedad de resultados. Algunos de ellos tienen que ver con el tiempo de la simulación y también el tiempo entre dos fotogramas de una misma secuencia. En este caso, la frecuencia deseada es de 25 fotogramas por segundo, que es la utilizada en la mayoría de televisores actuales.

2.1.1. Convirtiendo de binario a OBJ

La salida del simulador dispone de dos diferentes formatos que se eligen en el fichero mencionado anteriormente. Los dos formatos posibles son en ASCII o en binario. Para este proyecto se ha elegido la salida en formato binario. Este fichero contiene la información de todos los vértices de la matriz y también su energía, que se utiliza dentro del simulador para poder continuar la secuencia, pero en este proyecto de desechará.

El siguiente paso es convertir estas matrices al mencionado fichero OBJ para hacerlas compatibles con el entorno de renderizado. Este paso se realiza utilizando Matlab.

La función de conversión carga todos los archivos que hay en el directorio actual y que sigan el formato especificado, que en este caso es "EP_xxxxx.bin" ya que es el nombre por defecto que devuelve el simulador. Entre otras cosas, el conversor también añade las líneas que van a determinar el grupo al que pertenece el objeto y también su material. El fichero de materiales se llama "flow.mtl" y el material asignado deberá estar contenido en este fichero. Este fichero contiene los valores ambiental, difuso y especular del material, así como el valor "illum" que determinará el sombreador a utilizar en el entorno de renderizado. En este caso, el material a utilizar será "seawater" para las matrices que representan el agua.

La función de Matlab desde el primer momento se encarga de convertir todos los ficheros que se encuentren con el formato explicado anteriormente, ya que se necesitarán más adelante en el contexto del proyecto. El nombre de los ficheros sigue el mismo nombre por defecto que anteriormente, de manera que los ficheros "EP_xxxxx.bin" se transforman en "EP_xxxxx.bin.obj".

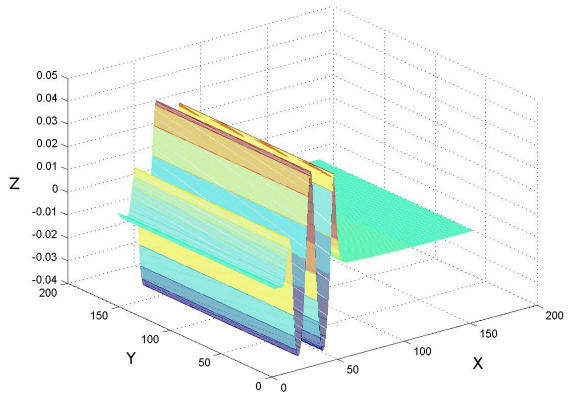

Figura 2.2: En esta figura se puede ver la geometría de una de las mallas del simulador visualizada en Maltab. La representación está realizada en unidades genéricas de longitud. El color de la malla está determinado por su magnitud en el eje Z.

Además, se ha incluido una función que dibuja la malla en Matlab para comprobar la corrección del mismo. Un ejemplo de esta visualización en Matlab se

puede ver en la figura 2.2.

2.2. El entorno de renderizado

Esta sección detalla en qué consiste el entorno de renderizado facilitado por el departamento DTU Compute. Este entorno tiene algunas funciones básicas implementadas, aunque su parte principal, que son los sombreadores, no están implementados. Todas las referencias al entorno de rendering que se encuentren fuera de esta sección han tenido que ser implementadas, mientras que las funciones que ya estaban incluídas se detallan a continuación.

2.2.1. La ecuación de render

En primer lugar, es necesario definir en qué consiste el proceso de renderizado. Renderizar el es proceso de generar una imagen mediante el cálculo de de la iluminación de una escena en tres dimensiones. Para determinar la iluminación en cada punto de la escena se utiliza la ecuación de render [Kaj86a]:

$$L(\mathbf{x}, \omega_\mathrm{o}) \;=\; L_e(\mathbf{x}, \omega_\mathrm{o}) \;+\; \int_\Omega f_r(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})\, (\omega_\mathrm{i} \cdot \mathbf{n})\, \mathrm{d}\,\omega_\mathrm{i}$$

El resultado de la ecucación es la radiancia $L(x,\omega_o)$, la cual viene determinada en función de la posición x y la dirección ω_o, y es el resultado de sumar la radiancia emitida por la superficie $L_e(x,\omega_o)$ y la radiancia indicente $L(x,\omega_i)$ en el punto x procedente de todas las direcciones, donde ω_i es la dirección de incidencia. El término $(n \bullet \omega_i)$ representa la atenuación según el ángulo de incidencia y el término $f_r(\mathbf{x}, \omega_i, \omega_o)$ representa el BRDF (Bidirectional Reflectance Distribution Function) en el punto x que determina la forma en la que es reflejada la luz en la superficie.

2.2.2. Raytracing

Este proyecto utiliza un entorno de renderizado basado en raytracing [App68]. Como crear un *raytracer* desde cero llevaría más tiempo que el propio proyecto, se ha proporcionado el utilizado en la asignatura Phisically Based Rendering del departamento DTU Compute, el cual contiene algunas funciones básicas ya implementadas aunque no incluye ningún sombreador entre otras cosas.

La técnica de raytracing consiste en la emisión de rayos desde la cámara a través de cada uno de los píxels de la imagen. Cuando los rayos encuentran un objeto, si éste tiene propiedades de reflexión o refracción, se trazarán dichos rayos desde este nuevo punto, y se continuará haciendo recursivamente con cada intersección con un nuevo objeto. Además, se trazará un rayo hacia la fuente de luz, el cual, si no atraviesa ningún otro objeto, será sombreado calculando la cantidad de luz recibida y su ángulo, además de con los valores de reflexión y refracción, mientras que si el rayo atraviesa algún objeto, el valor se calculará solamente con los valores de reflexión y refracción al estar en sombra.

En este proyecto, se ha establecido la cantidad máxima de divisiones de rayos en 10. A partir de ese valor, se aplicará path tracing. Lo que hace esta técnica es seguir los rebotes de uno posibles caminos en vez de hacerlo de todos, lo que hace que pueda aparecer ruido en las imágenes, aunque el proceso será más rapido a partir de ese punto. Para ello, se calculará un valor aleatorio para elegir o bien el rayo reflejado, o bien el refractado. El proceso se ha configurado para que termine después de 20 rebotes.

Entre las utilidades que incluye el entorno de renderizado cabe mencionar la de importar ficheros OBJ y guardar imágenes PNG de los resultados que serán utilizadas en este proyecto. Además, el entorno utiliza internamente una estructura de árbol BSP (Binary Space Partition) [SS92] para almacenar la geometría en memoria.

Dentro del entorno, el usuario puede mover la cámara con el ratón, guardar y cargar la vista y la posición de la cámara e incrementar o decrementar el número de rayos por píxel que serán utilizados. Además, aunque el proyecto permite utilizar cualquier número de luces, solo se va a utilizar una luz direccional que representará el sol.

Además, el entorno también controla diferentes tipos de visualización, cuyos sombreadores están inicialmente vacíos, como solo iluminación directa, oclusión ambiental, path tracing o photon mapping, aunque no todos se van a utilizar en este proyecto. Además, aunque todas estas técnicas se pueden implementar en el entorno, hay que saber antes de nada cuales serán utilizadas y desechar el resto para no implementarlas innecesariamente.

2.2.3. Materiales

El entorno también incluye un fichero de materiales llamado “media.mpml”. Si en el fichero “flow.mtl” descrito anteriormente se encuentra algún material coincidente, se aplicarán las propiedades descritas en ambos ficheros. En contreto,

este fichero contiene un material "seawater" que incluye más propiedades sobre el agua. En este fichero también se podría incluir nuevos tipos de agua ya que los diferentes océanos tienen ligeras variaciones en su aspecto.

2.2.4. El sol y el cielo

Habitualmente, las escenas generadas por ordenador suelen ser en entornos cerrados, sin embargo, este proyecto genera una escena al aire libre, de manera que en lugar de usar un color de fondo para todo el cielo, se va a añadir un método para calcular los colores del cielo. Además, este color también afectará al aspecto del agua al ser reflejado por ella.

Un modelo muy utilizado hoy en día, y que además es computacionalmente asequible, es modelo de Preetham, Peter Shirley y Brian Smits de la Universidad de Utah [PSS99]. Este modelo simplifica enormemente los cálculos para obtener la luz atmosférica que alcanza cada punto de la escena y aporta un gran realismo a la misma. El modelo utiliza las coordenadas reales de la Tierra, así como la fecha y la hora.

El modelo simplifica los cálculos necesarios debidos a la dispersión de la luz en la atmósfera teniendo en cuenta que en la dirección que mira el observador pueden llegar rayos que han sido reflejados en distintos puntos de la atmósfera. Un ejemplo se puede ver en la figura 2.3. Además el modelo simplifica algunos parámetros de la atmósfera que habitualmente son desconocidos o muy dificiles de calcular.

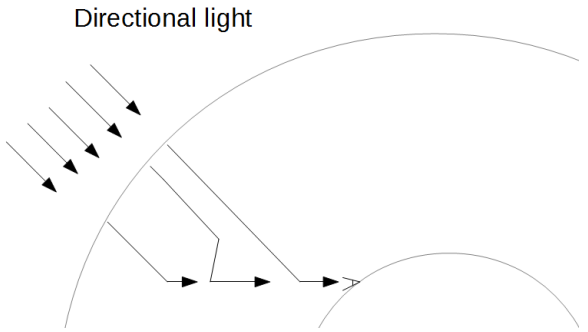

Figura 2.3: Diagrama de la dispersión en el cielo.

Este modelo también incluía parte de su estructura en el entorno de renderizado facilitado para realizar este proyecto, aunque se han tenido que realizar pequeños

ajustes. Para este proyecto, la fecha elegida ha sido un día de otoño a las 12.00 y se ha localizado en Dinamarca. Estos valores pueden ser modificados en cualquier momento en el entorno de renderizado.

2.3. Adaptación del entorno al simulador

Aunque el entorno de renderizado permite importar objetos en formato OBJ, es necesario realizar algunos ajustes para su correcta visualización. Por ello, después de cargar el objeto correspondiente en memoria, se realizan los siguientes cambios sobre el mismo.

El simulador, por defecto, no incluye el fondo marino, y dado que es importante para la visualización, se ha procedido a incluirlo dentro del entorno de renderizado. De esta manera, se colocará un cuadrilátero inclinado por debajo de la malla de agua. Esta opción no es del todo precisa y por eso lo deseable es obtener el fondo marino directamente del simulador. De hecho, las últimas versiones del simulador ya lo generan por defecto.

Con la actual configuración, la luz podría llegar al fondo marino sin pasar por la superficie. Este fenómeno se puede apreciar en detalle en la figura 2.4.

Para arreglar esta cuestión, se ha creado una caja que rodee tanto el agua como el fondo marino, creando algo similar a una piscina. Esta caja debe abarcar desde el punto más alto de la ola hasta el punto más bajo del fondo marino.

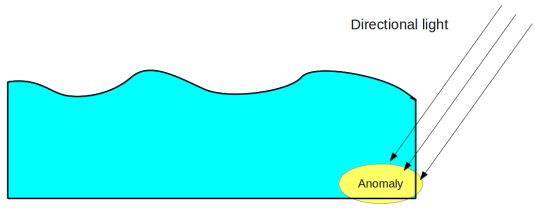

Figura 2.4: Este diagrama muestra por qué es necesario cubrir los laterales del agua. Si no existiesen el agua alcanzaría el fondo marino sin pasar por la superficie. Al añadir estos cuadriláteros sigue habiendo una anomalía, ya que la escena será más oscura en los bordes, pero el resultado será mucho más preciso que anteriormente.

Usando este método, todavía hay un efecto indeseado, ya que al acercarse a las

esquinas, el aspecto del agua será más oscuro al llegar menos rayos al fondo marino dependiendo del punto y del ángulo del sol. Además, el lado que mira directamente al sol acumulará una cierta cantidad de fotones que no le correspondería (Esto será detallado más adelante en el apartado de photon mapping). Este efecto puede ser mejorado creando mallas más grandes o creando playas suaves en la intersección entre el fondo marino y la superficie del agua.

Finalmente, se ha añadido un plano que representa el suelo. Este suelo se ha colocado más abajo de lo que le correspondería de manera que el agua estaría flotando. Esto se ha hecho para que la visualización del horizonte sea mas coherente a como es en realidad, aunque esto crea una sombra en el suelo. Este fenómeno también se puede evitar creando mallas más grandes como se ha explicado anteriormente.

2.4. Producción de vídeo

La producción de vídeo se gestiona utilizando la línea de comandos al ejecutar la aplicación. En estos argumentos se define cual es la primera y la última iteración a renderizar, y también el periodo de tiempo entre cada una de ellas.

Si el número de la primera iteración es menor que el último, se procederá a un renderizado en cadena, tomando progresivamente las diferentes simulaciones hasta que termine la última. El nombre de las imágenes resultantes será "EP_xxxxx.bin.obj.png" siguendo el mismo formato que en todos los pasos anteriores.

Para poder crear una escena de vídeo, hay que ejecutar el programa, colocar la cámara en el lugar deseado, y posteriormente, pulsar "4" y "R" para comenzar el renderizado. Durante el proceso, se almacenarán en disco los fotogramas renderizados, sin embargo, la visualización de la escena en la aplicación no se actualizará hasta que se haya terminado el último fotograma.

Finalmente, para poder montar las imágenes y generar secuencias de vídeo, es necesario utilizar una aplicación externa como podría ser Windows Movie Maker.

2.5. Diagrama de clases

Esta sección muestra un diagrama de clases simplificado del entorno de renderizado. En dicho diagrama se han coloreado de verde todas las clases modificadas en este proyecto, aunque la que ha sufrido la mayoría de los cambios ha sido la clase *RenderEngine*. El diagrama de clases se puede ver en la figura 2.5. En este diagrama se han simplificado los sombreadores dejandolos como una única clase, aunque este diagrama se desglosará en el capítulo 3 que trata sobre todos los sombreadores implementados.

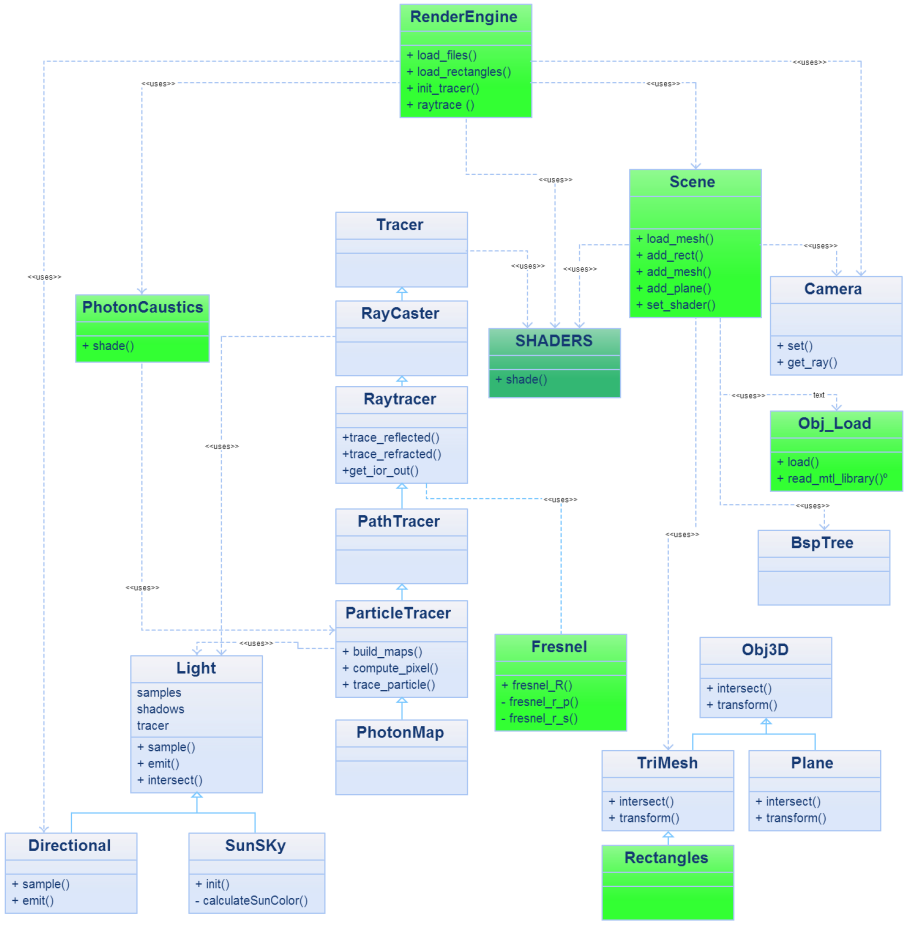

Figura 2.5: Diagrama de clases del entorno de renderizado.

Capítulo 3

Sombreado

Este capítulo explica todas las propiedades que afectan al aspecto del agua, por ello, es importante saber qué técnicas permiten representar dichas propiedades. A su vez, también se explican las dificultades encontradas.

El término que se usa en el ámbito internacional es *shading*, que en este documento se ha traducido como proceso de sombreado, aunque no se refiere a las sombras generadas por los objetos como tal, sino al proceso de calcular la iluminación que incide sobre cada punto de la geometría.

A continuación se explican los diferentes sombreadores utilizados, siendo algunos de ellos implementados unos sobre otros hasta completar todas las propiedades que afectarán al aspecto del agua. Todas las técnicas descritas en este capítulo se han implementado sobre el entorno de rendering explicado en el capítulo anterior[Lew93] [Kaj86b].

En cuanto a los materiales, sus propiedades están definidas en el fichero "flow.mtl". En este fichero se han definido los materiales que van a ser utilizados en todo el proyecto, pero es posible definir tantos nuevos materiales como se desee.

3.1. Lambertian

Este sombreador es el más sencillo que se va a utilizar en este proyecto, pero es necesario para, entre otras cosas, el fondo marino.

El sombreador se utiliza para materiales que tienen una superficie puramente difusa. En este caso, la cantidad de luz reflejada por la geometría depende únicamente del ángulo entre la luz y la normal del objeto en el punto donde incide la luz. El BRDF (Bidirectional reflectance distribution function) se puede apreciar en la figura 3.1.

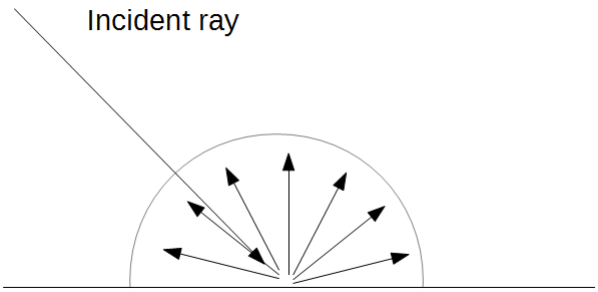

Figura 3.1: Lambertian BRDF

La formula que determina el color para este tipo de superficies es la siguiente:

$$L_o = (\omega \bullet n) * C * V * I_L$$

Para calcular la cantidad de luz en ese punto L_o, se calcula el producto escalar entre la dirección ω del rayo y la normal n de la geometría en ese punto, a lo que hay que añadir el color C y la intensidad de la luz I_L. El parámetro V se utiliza para determinar si el punto se encuentra al alcance la luz o no, por lo que será 1 si está en ese caso, y 0 si está en una región de sombra.

A modo de ejemplo, la escena se ha renderizado utilizando este material para el agua. El resultado se puede ver en la figura 3.2.

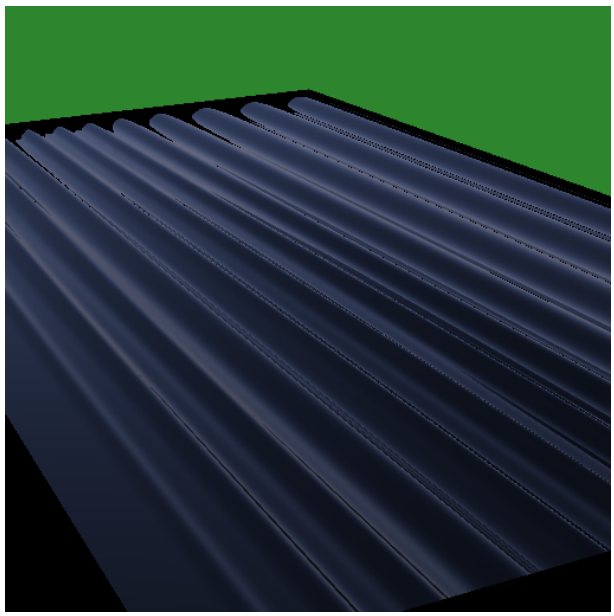

Figura 3.2: Agua renderizada utilizando el sombreador Lambertian

3.2. Sombreador transparente

Este sombreador está construído independientemente del anterior, y se ha creado para poder visualizar materiales transparentes, y sirve como primera aproximación para poder renderizar el agua, ya que los siguientes sombreadores se implementarán sobre este.

Para ello, se han tenido que implementar las ecuaciones de Fresnel. Estas ecuaciones sirven para determinar la reflectividad del medio en cada uno de sus puntos, dependiendo de los índices de refracción y el ángulo de entrada. La reflectividad R determinará la cantidad de luz reflejada mientras que la restante (1-R) será refractada [Ska06]. Las ecuaciones de Fresnel se pueden ver en la figura 3.3, donde ya han sido simplificadas utilizando la ley de Snell, la cual relaciona los diferentes índices de refracción con los ángulos de entrada y de refracción en el medio.

Posteriormente, este sombreador traza dos rayos, el reflejado y el refractado, tal y como se ha mencionado en la introducción de este capítulo, siendo combinados en función de la reflectividad obtenida [JB02]. El diagrama de los rayos reflejado

$$R = \frac{R_s + R_p}{2}$$

$$R_s = \left| \frac{n_1 \cos\theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2}}{n_1 \cos\theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2}} \right|^2$$

$$R_p = \left| \frac{n_1 \sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2} - n_2 \cos\theta_i}{n_1 \sqrt{1 - \left(\frac{n_1}{n_2}\sin\theta_i\right)^2} + n_2 \cos\theta_i} \right|^2$$

Figura 3.3: Ecuaciones de Fresnel para calcular la cantidad de luz reflejada y refractada.

y refractado se puede ver en la figura 3.4.

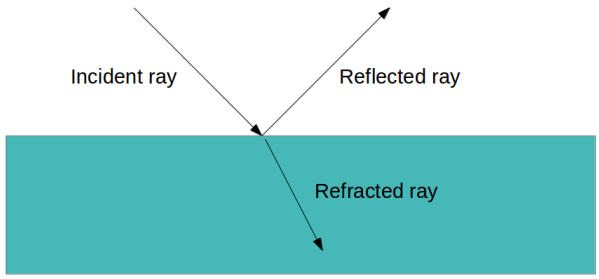

Figura 3.4: Este diagrama muestra los rayos reflejado y refractado, que son usados en éste y otros sombreadores.

3.3. Photon mapping

En este proyecto, uno de los aspectos que se ha tenido en cuenta desde el primer momento ha sido el fondo marino, que afectará significativamente al aspecto del agua, y una de las formas en las que afectará al aspecto del agua es debido a las causticas que se puedan formar en el fondo.

Sin embargo, raytracing es un algoritmo que no encuentra una solución óptima para algunas características muy concretas, como son las cáusticas. Ello es de-

bido a que al trazar los rayos desde la cámara, y despues de sucesivos rebotes, el rayo termina en la luz de la escena. De esta manera, hay muchos caminos que la luz en una escena real está recorriendo, pero que no son apreciables al usar este método. Concretamente, las cáusticas aparecen al converger la luz después de atravesar superficies refractivas, y se necesita de una técnica más potente para poder visualizarlas. En la figura 3.5 se puede ver cómo se producen las cáusticas.

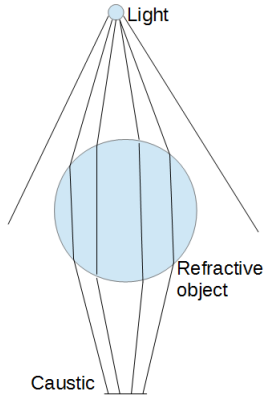

Figura 3.5: Esta figura muestra la formación de cáusticas a través de un objeto transparente.

Por ello, se ha utilizado la técnica de Photon mapping para, en caso de que existan, poder obtener las causticas que se puedan formar [NJC00]. Este método consiste en emitir fotones (partículas indivisibles de energía) desde la fuente de luz, y seguir su camino por la geometría hasta que encuentren una superficie donde se puedan almacenar.

Photon mapping es un algoritmo que consta de dos pasadas. En la primera, se emite una serie de fotones desde la fuente de luz y se calculan sus rebotes hasta que alcanzan una superficie difusa donde se puedan almacenar. En el caso de tener varias opciones como reflexión o refracción, se elige aleatoriamente una de ellas. Una vez terminado el proceso, se obtiene un mapa de fotones. Finalmente, se realiza una estimación de la energía almacenada para determinar la cantidad de luz sobre la superficie difusa en función de la cantidad de fotones en un área determinada. En la figura 3.6 se puede ver la ecuación para estimar la radiancia, que es una aproximación a la ecuación de render.

Este método, en lugar de dividir un fotón en varios caminos, elige aleatoriamente

$$L(\mathbf{x}, \vec{\omega}) \approx L_e(\mathbf{x}, \vec{\omega}) + \sum_{p \in \Delta A} f_r(\mathbf{x}, \vec{\omega}'_p, \vec{\omega}) \frac{\Delta \Phi_p(\mathbf{x}, \vec{\omega}'_p)}{\Delta A}$$

Figura 3.6: Esta figura muestra la ecuación para la estimación de la radiancia, que es una aproximación a la ecuación de render.

uno de ellos, de manera que el método es más preciso según se aumenta la cantidad de fotones emitidos, lo que, por otra parte, lo hace más lento. De esta manera, al incrementar el valor, será más probable que los diferentes caminos sean tomados por los diferentes fotones haciéndolo así más preciso.

La formación de las causticas dependerá de la forma de las olas y también de la distancia desde la superficie hasta el fondo, ya que solo se formarán donde converjan una gran cantidad de fotones.

Como se ha explicado anteriormente, el entorno de renderizado incluye una opción para visualizar el resultado de photon mapping, aunque el sombreador ha tenido que ser implementado (si no se hace, el visualizador simplemente no muestra nada). El motivo de la elección de Photon mapping para visualizar las cáusticas se debe a que este algoritmo está construido sobre raytracing, que es la técnica principal de este proyecto.

Este método estará activado siempre que al renderizar se utilice el visor número "4" en el entorno de desarrollo. El número de fotones se puede ajustar en la aplicación, así como el número de fotones usados en la estimación. En este proyecto, estos valores son 7.500.000 fotones y 200 para la estimación.

En la figura 3.7 se pueden ver tanto el mapa de fotones como el resultado de aplicar photon mapping al agua transperente.

3.4. Absorción

El siguiente efecto que se va a utilizar tiene que ver con la profundidad del agua. De esta manera, el agua será mas oscura cuanto más profunda sea, llegándose a un punto en el que el fondo marino no llegue a ser visible.

En este sombreador, se planteó la posibilidad de utilizar scattering [GSMA08] [DGJ08]. Esta técnica lo que haría sería reflejar o refractar el rayo en diferentes puntos dentro de un volumen. A esta acción se le denomina evento de scattering,

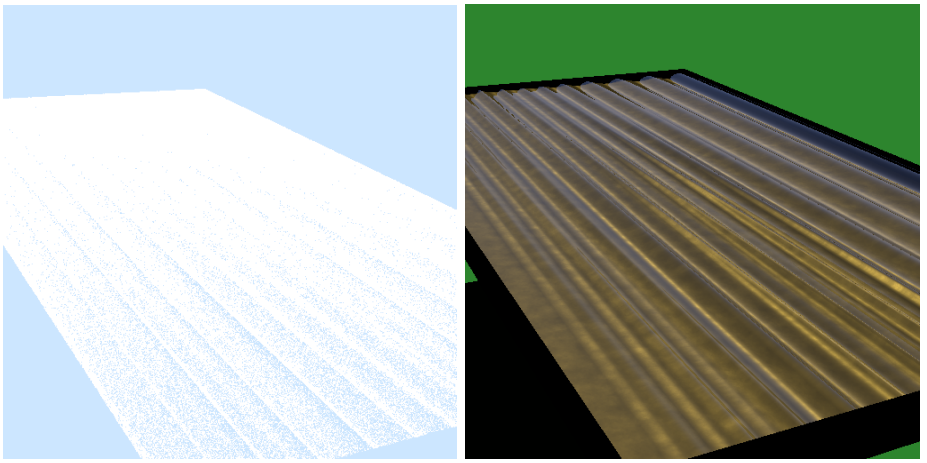

Figura 3.7: Izquierda: Visualización de los mapas de fotones. Derecha: Resultado del renderizado utilizando un sombreador transparente combinado con photon mapping. Las cáusticas se pueden ver perfectamente en el fondo marino.

y se producirían cuando se encuentre una partícula dentro del volúmen. Sin embargo, aunque en este proyecto se trabaje con fluídos, estos no pertenecen a un volúmen, si no que se realiza utilizando diferentes superficies las cuales forman una geometría cerrada, de manera que se optó por descartar esta técnica. Además, utilizar scattering hubiera supuesto un tiempo de renderizado mucho mayor debido a los múltiples eventos que ocurrirían dentro del volúmen.

En su lugar, se va a aplicar absorción. El término de absorción es la probabilidad de que la luz sea absorbida por el medio que está atravesando. En este caso, lo que se hace es trazar un rayo en la dirección de refracción y se mide la distancia desde la superficie hasta el fondo, siguiendo la dirección de dicho rayo. De esta manera, se puede aplicar el coeficiente de absorción en función de la longitud del rayo [EC05]. Este sombreador forma parte de una nueva clase, sin embargo, se utiliza sobre el de materiales transparentes al cuál se le añade el término de absorción. La figura 3.8 muestra la escena usando absorción de manera aislada (sin photon mapping).

Es a partir de este punto donde adquiere sentido la "piscina" que se ha creado envolviendo a los objetos, ya que la única manera de que entre luz en el fondo es atravesando la superficie. Además, este sombreador se ha construido sobre el transparente, ya que el termino de absorción se aplica sobre dicho sombreador. De esta manera, la cantidad de luz que llega al fondo es muy pequeña, y a partir

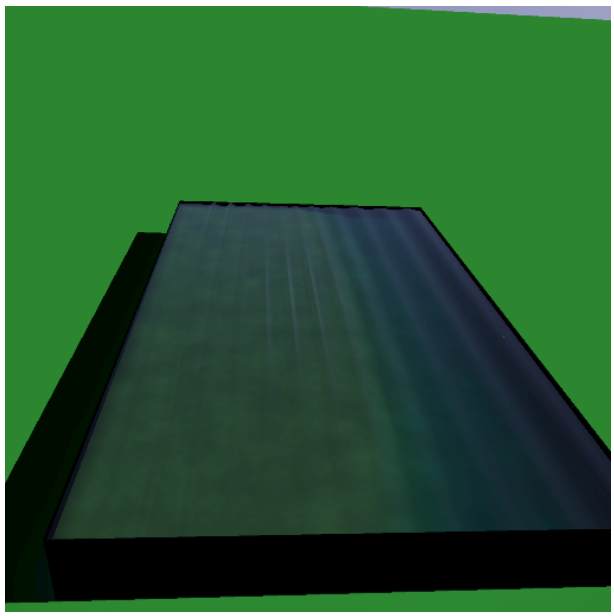

Figura 3.8: Esta figura muestra como el color del agua es afectado por la absorción. En la parte izquierda de la imagen se puede ver como la profundidad es menos y el color resultante es mas claro. En la parte derecha, sin embargo, el color es más oscuro debido a que el agua es más profunda.

de este punto casi toda la luz que alcance el fondo será a través de fotones, cuyas causticas serán visibles desde el exterior si la absorción lo permite.

3.5. Modelo de reflexión de Phong

Por último, para contribuir un poco más al aspecto del agua, se ha implementado el modelo de reflexión Phong, que no ha de ser confundido el modelo de sombreado de Phong. Este modelo contribuye a la reflexión del agua, que reflejará la luz del sol cuando el ojo, la superficie del agua, y el sol, estén en el mismo plano [Pho75]. La ecuación para aplicar la reflexión de Phong se encuentra en la figura 3.9.

Esta ecuación solo aplica la componente especular de la luz, debido a que la iluminación directa ya se ha calculado anteriormente. De esta manera se consigue

$$L_r = k_s \cos^s \alpha \, L_i \cos \theta$$

Figura 3.9: Esta figura muestra la ecuación de la reflexión de Phong.

que se refleje el sol en la superficie del agua en caso de que la cámara esté en el lugar apropiado. De esta manera, la luz reflejada L_r será la componente especular k_s del objeto, multiplicado por el factor $\cos(\alpha)^s$, donde s es el brillo y α es el coseno del ángulo entre el vector que une el punto de la geometría con la cámara y el vector normalizado del rayo reflejado, la luz emitida L_i y el coseno del ángulo θ, que es el ángulo formado por el vector que une el punto de la geometría con la cámara y la normal del la geometría.

La figura 3.10 muestra la escena utilizando la reflexión de Phong junto con el resto de propiedades descritas hasta el momento.

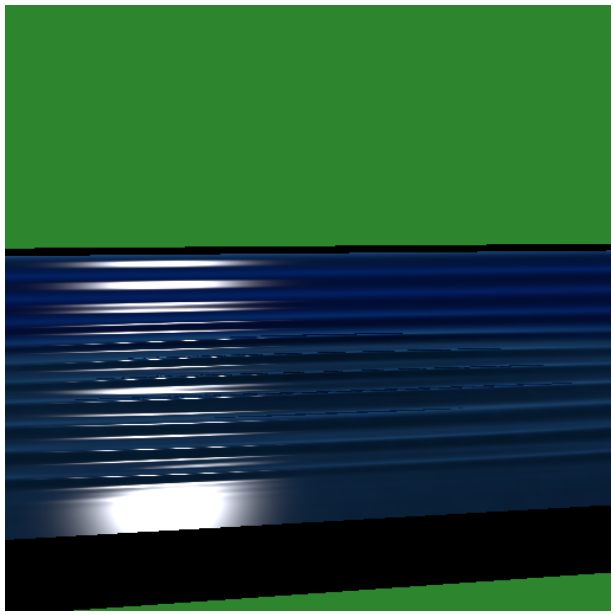

Figura 3.10: Esta figura muestra como el sol es reflejado en la superficie del agua usando la reflexión de Phong.

3.6. Comentarios finales

Como se ha explicado anteriormente, en la versión final de la aplicación el usuario puede elegir entre los diferente sombreadores, y esto se hace en el fichero "flow.mtl". Dentro de este fichero es donde elige el sombreador definiendo el valor "illum" apropiado. En este caso, para el agua se ha utilizado el valor 15. Este sombreador combina absorción, photon mapping y reflexión de Phong. Este sombreador ha sido utilizado, por ejemplo, en la figura 3.11.

Otro sombreador utilizado es el transparente, aunque este sólo se ha utilizado en los ejemplos. En este caso el valor "illum" tiene que ser 4 y también utiliza photon mapping. Este sombreador ha sido utilizado en la imagen de la derecha de la figura 3.7.

Por último, el sombreador difuso ha sido utilizado como ejemplo para el agua en la figura 3.2 aunque ha sido utilizado para el fondo marino en todas las demás imágenes.

Además, el modelo del cielo y el sol se ha utilizado en todas las imágenes y no está vinculado a los sombreadores.

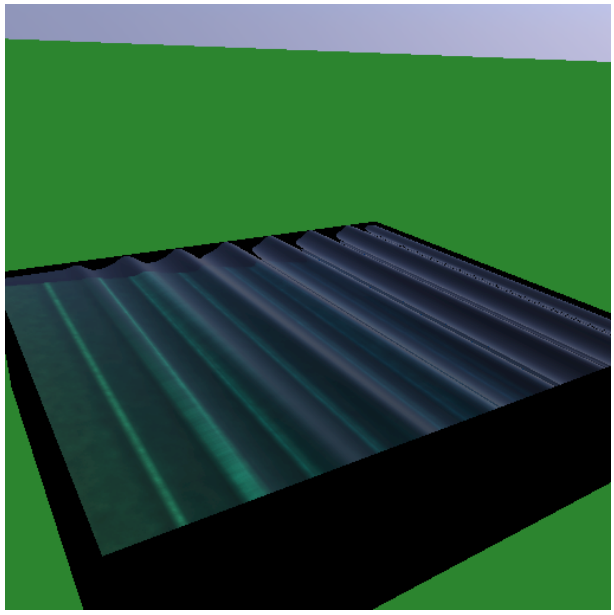

Figura 3.11: Esta figura muestra las cáusticas y absorción en el agua

El diagrama de clases con la estructura de todos los sombreadores se pueden ver en la figura 3.12. Como se ha explicado anteriormente, el entorno incluía una estructura básica de algunos sombreadores, aunque ninguno estaba implementado. En el diagrama se han señalado en color verde los sombreadores implementados, en los cuales se ha incluído el método `shade` que se hereda desde el sombreador más básico `Shader.h`.

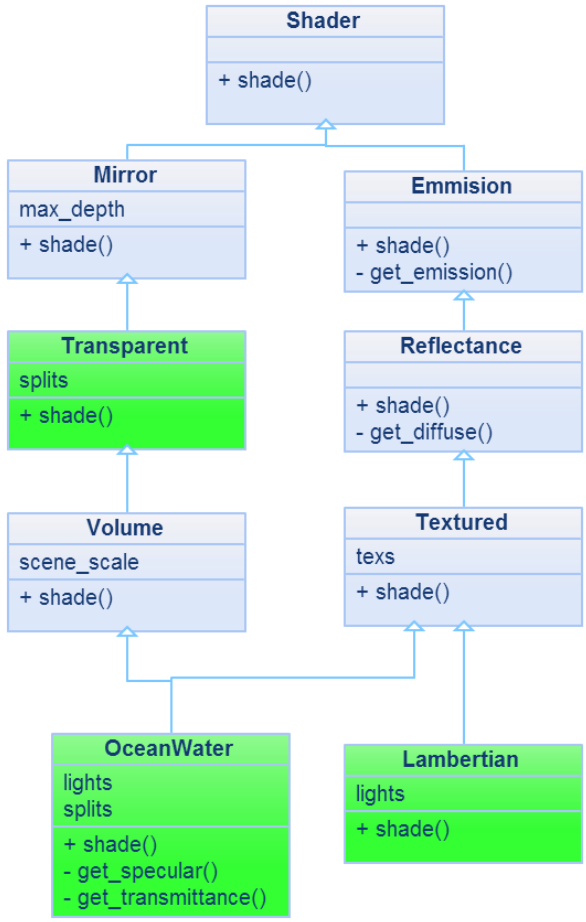

Figura 3.12: Diagrama de clases de los sombreadores

Capítulo 4

Resultados

Una vez que se ha terminado la implementación, se han llevado a cabo varias simulaciones cuyo objetivo es estudiar el tiempo consumido y generar secuencias de vídeo. Dos de los ejemplos se han configurado para generar dos secuencias de vídeo, mientras que otra se ha realizado con el objetivo de obtener una sola imagen aunque con mucho más nivel de detalle.

Para llevar a cabo las simulaciones, se ha determinado la frecuencia en 25 imágenes por segundo que es la que se usa actualmente en las televisiones europeas. De esta manera, hay que configurar el simulador para obtener un fotograma cada 0.04 segundos. En el caso de las simulaciones para una sola imagen, se ha utilizado más de un rayo por píxel, que es una opción que, como se ha explicado anteriormente, viene implementada en el entorno de renderizado.

4.1. Ola lineal

Esta simulación da como resultado una ola en dos dimensiones, de manera que para transformar a 3 dimensiones, simplemente se ha extendido en la dimensión restante. Al ser una ola en solamente dos dimensiones, se espera que sea computacionalmente sencilla. Esta simulación va a generar una secuencia de 600 fotogramas y creará un vídeo de 24 segundos. La figura 4.1 muestra el tiempo

requerido por cada uno de los procesos y también el tiempo medio por fotograma. En este caso, el tamaño de la malla que forma el agua es de 259 x 2 vértices en cada dirección.

LINEAR	600 frames	Average
Simulation	3 minutes	0,3 seconds
Conversion	42 seconds	0,07 seconds
Rendering	83,3 hours	8,3 minutes

Figura 4.1: Esta tabla muestra los tiempos para el ejemplo de una ola lineal

En este caso, la simulación ha durado 0,3 segundos por cada fotograma, de manera de que el tiempo total ha sido de 3 minutos.

El tiempo de conversión también ha sido significativo, aunque este proceso ha sido mucho más rápido. En este caso el tiempo ha sido de 0,07 segundos por cada fotograma, mientras que el tiempo total ha sido de 42 segundos.

La geometría se puede ver en la figura 4.2 tanto como se ve en Matlab como después de renderizada. Las salidas de esta simulación también han sido utilizadas en otras partes de la memoria.

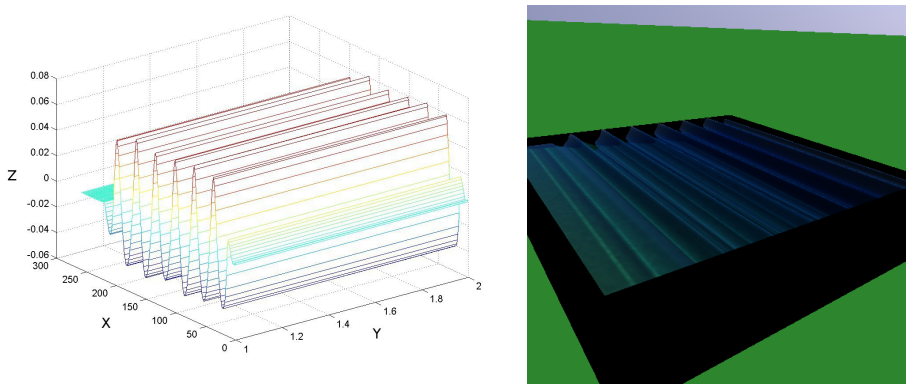

Figura 4.2: La imagen de la izquierda representa la geometría de la malla visualizada en Matlab, cuyo color es determinado por la coordenada Z. La imagen de la derecha es la visualización de la misma malla, ésta vez visualizada después de renderizar.

Finalmente, el tiempo de renderizado ha sido de unos 8 minutos de media, de manera que el tiempo para los 600 fotogramas ha sido de unas 83 horas. En el apéndice A se pueden ver algunos de los fotogramas pertenecientes a esta secuencia.

4.2. Ola no lineal

Esta simulación genera una ola en 3 dimensiones y consta nuevamente de 600 fotogramas que representarán 24 segundos de vídeo. En este caso, al ser una simulación en 3 dimensiones, se espera que la simulación sea más lenta que en el caso anterior. La figura 4.3 muestra los tiempos obtenidos en los tres pasos que requiere el proceso. En este caso, el tamaño de la malla que forma el agua es de 259 x 19 vértices en cada dirección.

NON LINEAR	600 frames	Average
Simulation	110 minutes	11 seconds
Conversion	5,5 minutes	0,55 seconds
Rendering	35,6 hours	3,5 minutes

Figura 4.3: Esta tabla muestra los tiempos obtenidos para el ejemplo de la ola no lineal.

En este caso, la conversión también ha sido más lenta que anteriormente debido al mayor número de vértices que procesar, aunque este paso ha sido nuevamente el más sencillo de los tres.

Finalmente, el renderizado de la secuencia ha tardado una media de 3,5 minutos por fotograma siendo el tiempo total de 35 horas. En este caso, el tiempo de una sola imagen ha tomado entre 180 segundos para el caso mejor y 380 segundos para el caso peor.

Aunque esta ola es una ola en 3D, en la visualización después de renderizar es muy dificil de apreciar ya que avanza en una sola dirección, sin embargo, como se puede ver en la figura 4.4, en la visualización en Matlab se pueden observar sus diferencias. En el apéndice A se pueden ver algunos de los fotogramas pertenecientes a esta secuencia.

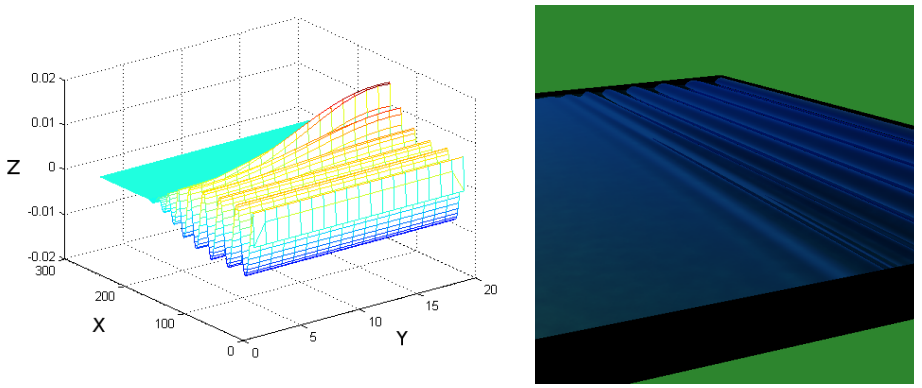

Figura 4.4: La imagen de la izquierda representa la geometría de la malla visualizada en Matlab, cuyo color es determinado por la coordenada Z. La imagen de la derecha es la visualización de la misma malla, ésta vez visualizada después de renderizar.

4.3. Simulación de la empresa Force Technology

La siguiente simulación es cortesía de la empresa Force Technology, que se ha encargado de su simulación, de manera que se procederá solo a su transformación y renderizado. La simulación consiste en la estela dejada por un barco en la superficie del agua. El barco no tiene casco pero no es necesario para visualizar el aspecto del agua.

Las mallas utilizadas en esta sección han sido creadas utilizando el mismo simulador, aunque para poder obtener la información de esta simulación ha sido necesario realizar cambios menores en la función de Matlab. La simulación proporcionada se muestra en la figura 4.5 tal cual se visualiza en Matlab.

Como esta simulación representa solamente un fotograma, se ha configurado a 9 rayos por pixel para evitar el aliasing y obtener una imagen mucho más nítida. Los tiempos obtenidos para esta simulación se muestran en la figura 4.6, mientras que el resultado del renderizado se puede ver en la figura 4.7.

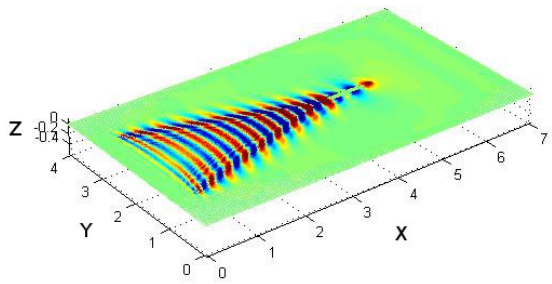

Figura 4.5: En esta figura se puede ver la geometría de una de las mallas del simulador visualizada en Maltab. La representación está realizada en unidades genéricas de longitud. El color de la malla está determinado por su magnitud en el eje Z.

FORCE	1 frame – 9 rays per pixel	Average (ray)
Simulation	-	-
Conversion	16 seconds	-
Rendering	53 minutes	5,8 minutes

Figura 4.6: Esta tabla muestra los tiempos obtenidos para renderizar la simulación proporcionada por Force Technology.

4.4. Comentarios

A lo largo de las simulaciones, ha sido necesario ajustar ciertos parámetros internos del entorno de renderizado para obtener resultados más precisos. Estos ajustes se han realizado para todas las simulaciones ejecutadas.

Una modificación ha consistido en escalar el sistema en el eje Z, debido a que en la mayoría de los casos, las olas no hubieran sido perceptibles. Además, también se ha modificado un parámetro que afecta al tamaño de la escena, el cual escala las distancias sin escalar la geometría, lo cual afecta a la absorción y a los fotones emitidos.

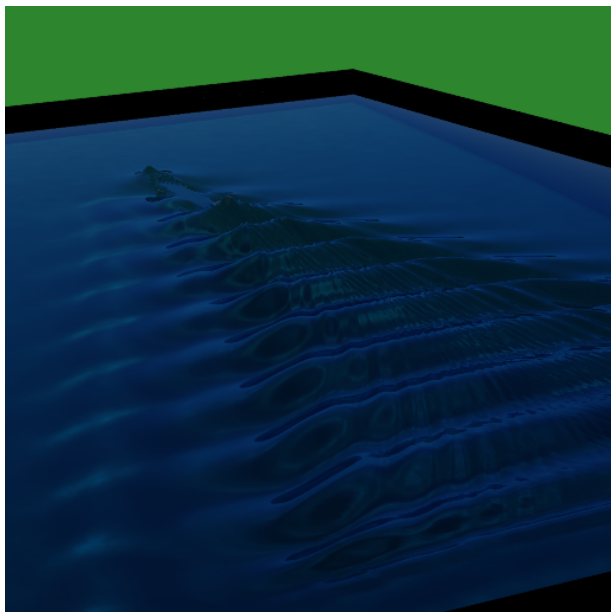

Figura 4.7: Renderizado de la simulación proporcionada por Force Technology

Como conclusión de este capítulo, cabe decir que las simulaciones han sido tan precisas y realistas como se esperaba, aunque no tan rápidas.

Capítulo 5

Conclusiones

Este proyecto ha desarrollado un completo entorno de renderizado para utilizar combinado junto con el simulador Ocean Wave [EKBL09] y que permite crear imágenes y vídeos del océano de una forma realista y precisa.

Este entorno de renderizado utiliza diferentes técnicas que, combinadas, permiten al usuario generar agua realísticamente, y además, añade sombreadores complementarios para renderizar de forma sencilla cualquier otro objeto que acompañe a la escena.

Además, como se ha visto en el capítulo de resultados, se puede importar cualquier tipo de escena siempre que sea compatible con el formato Wavefront OBJ. Además, con las funciones creadas en Matlab, se puede transformar fácilmente cualquier escena creada en el simulador a este formato, incluyendo la escena facilitada por Force Technology, cuyo formato era parecido aunque no igual que el simulador utilizado en este proyecto debido a sus diferentes versiones. Sin embargo, aunque el resultado ha sido muy preciso, éste no ha sido tan rápido como se hubiese deseado.

5.1. Limitaciones y mejoras futuras

Esta sección explica algunas limitaciones de este proyecto y diferentes formas de resolverlas.

Una gran limitación es el hecho de que la cámara no se puede mover a lo largo de un vídeo. La cámara se puede mover antes de empezar el proceso de renderizado para colocarla en el lugar deseado, sin embargo, en cuanto se empieza a renderizar el primer fotograma, la cámara ha de permanecer quieta hasta que termine el último.

La principal función del simulador es generar superficies de mar abierto, de manera que no se pueden incluir objetos que puedan afectar al aspecto del agua. Debido a que los objetos que rodean el agua tienen un gran impacto en su aspecto, sería una buena opción incluir este tipo de objetos en el futuro. Sin embargo, en el caso de que se introdujeran objetos de esta manera, se perdería coherencia con el simulador, ya que el simulador no habría tenido en cuenta su interacción con el agua. De esta manera, la única forma de tener objetos que mantengan la coherencia con el agua, sería incluirlos directamente en el simulador.

Como se ha comentado en secciones anteriores, algunas sombras son causadas debido a los límites laterales que se han creado alrededor de la superficie del agua. Aunque esta solución es mejor que no tener nada, no es del todo preciso, de manera que una solución puede ser crear suaves playas en los límites de la superficie.

Además, en esta aplicación, cada ejecución de renderizado es diferente y no se puede automatizar para lanzar diferentes ejecuciones. Esto quiere decir, que para cada ejecución hay que reconfigurar la posición de la cámara y ajustar parámetros interiormente para obtener un resultado preciso. Sin embargo , con las herramientas que tiene actualmente el entorno, sí que sería posible crear una secuencia de ejecuciones internamente, aunque previamente habría que guardar las posiciones de la cámara y dichos parámetros, para que al ejecutar, se puedan encadenar los diferentes procesos de renderizado.

Otra solución que iría mucho mas allá sería utilizar un fichero de entrada de la misma manera que hace el simulador, en el cual se puedan determinar todos los valores requeridos para cada proceso.

Por último, como se ha mencionado anteriormente, el renderizado ha sido más lento de los esperado. En el caso del simulador, existe una versión para GPU que es más rápida que la utilizada en este proyecto. Siguiendo la misma idea,

algunos sombreadores se pueden mover de la CPU a la GPU como es el caso de reflexión y refracción.

5.2. Conclusiones personales

Gracias a este proyecto, he aprendido a como gestionar grandes proyectos tal y como se hace en realidad, y a organizarlos y gestionarlos según lo planificado.

Este proyecto también ha supuesto un reto para mí porque desde el primer momento tuve que trabajar con un entorno de desarrollo muy grande, y antes de empezar a implementar todo lo necesario para este proyecto, tuve que aprender como estaba gestionado en su conjunto. Y gracias a ello, también he mejorado mi conocimiento en gráficos por ordenador y técnicas de renderizado.

Finalmente, el haber desarrollado este proyecto durante mi estancia en Dinamarca me ha permitido aprender como funcionan los departamentos fuera de la Universidad de Zaragoza.

5.3. Desarrollo del proyecto

Este proyecto ha tenido una duración de 7 meses. Los primeros 5 meses han concentrado el mayor esfuerzo, y durante este periodo, ha habido reuniones semanales en el grupo de gráficos del departamento DTU Compute para evaluar los progresos de los estudiantes que estábamos realizando algún proyecto. Además, en los momentos más importantes, también han tenido lugar reuniones privadas para mejorar ciertos aspectos del proyecto.

Por otra parte, al principio del proyecto hubo una reunión con la empresa Force Technology con sede en Kongens Lyngby, Dinamarca. Los temas tratados fueron la idoneidad de este proyecto para poder utilizar el desarrollo en sus instalaciones. Además se comprometieron a facilitar alguno de sus modelos para utilizar en este proyecto, el cual se puede ver en el capítulo 4.

A continuación se puede ver el diagrama de Gannt que describe la evolución de este proyecto.

Figura 5.1: Diagrama de Gannt del proyecto

Bibliografía

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[DGJ08] Srinivasa Narasimhan Diego Gutierrez, Henrik Wann Jensen and Wojciech Jarosz. Scattering. 2008.

[EC05] Xavier Pueyo Francisco J. Seron François X. Sillion Eva Cerezo, Frederic Pérez. A survey on participating media rendering techniques. 2005.

[EKBL09] A. P. Engsig-Karup, H. B. Bingham, and O. Lindberg. An efficient flexible-order model for 3d nonlinear water waves. *J. Comput. Phys.*, 228(6):2100–2118, April 2009.

[EKMG12] A. P. Engsig-Karup, Morten G. Madsen, and Stefan L. Glimberg. A massively parallel gpu-accelerated model for analysis of fully nonlinear free surface waves. *International Journal for Numerical Methods in Fluids*, 70(1):20–36, 2012.

[GSMA08] Diego Gutierrez, Francisco Seron, Adolfo Muñoz, and Oscar Anson. Visualizing underwater ocean optics. *Computer Graphics Forum (Proc. of EUROGRAPHICS)*, 27(2):547–556, 2008.

[JB02] Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph.*, 21(3):576–581, July 2002.

[JL04] Claes Johanson and Calle Lejdfors. Real-time water rendering. *Lund University*, 2004.

[Kaj86a] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[Kaj86b] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[Kry05] Yuri Kryachko. *Using vertex texture displacement for realistic water rendering*, volume 2. 2005.

[Lew93] Robert R. Lewis. Making shaders more physically plausible. Technical report, Vancouver, BC, Canada, Canada, 1993.

[NJC00] Henrik Wann Jensen Niels Jørgen Christensen. A practical guide to global illumination using photon maps. 2000.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[PSS99] A. J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 91–100, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Ska06] Johannes Skaar. Fresnel equations and the refractive index of active media. *Phys. Rev. E*, 73:026605, Feb 2006.

[SS92] Kelvin Sung and Peter Shirley. Graphics gems iii. chapter Ray tracing with the BSP tree, pages 271–274. Academic Press Professional, Inc., San Diego, CA, USA, 1992.

Apéndice A

Resultados adicionales

Esta sección contiene algunos fotogramas de los resultados obtenidos en el capítulo 4. La primera secuencia pertenece al ejemplo de la ola lineal, mientras que la segunda pertence al ejemplo de Whalin.

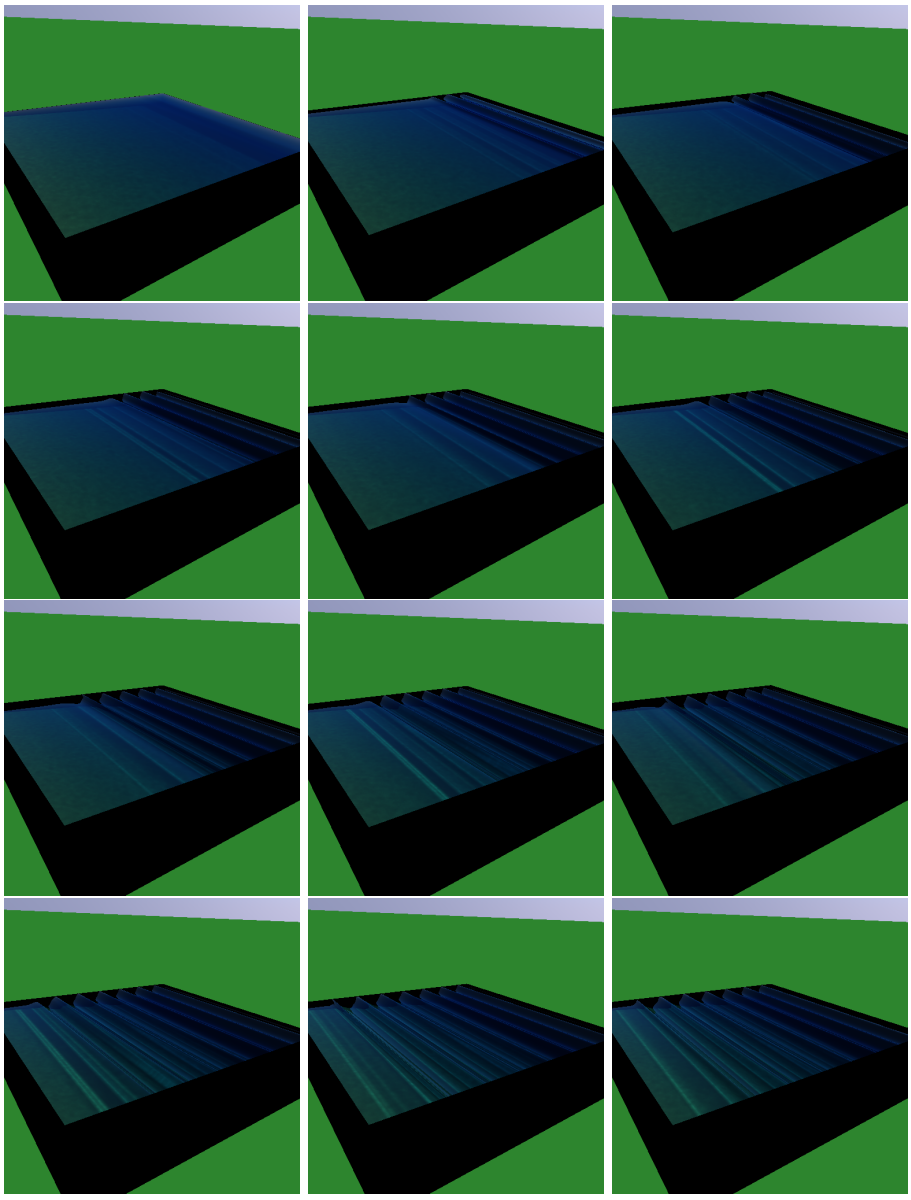

Figura A.1: Esta figura incluye algunos fotogramas pertenecientes a la secuencia de la ola lineal descrita en el capítulo 4

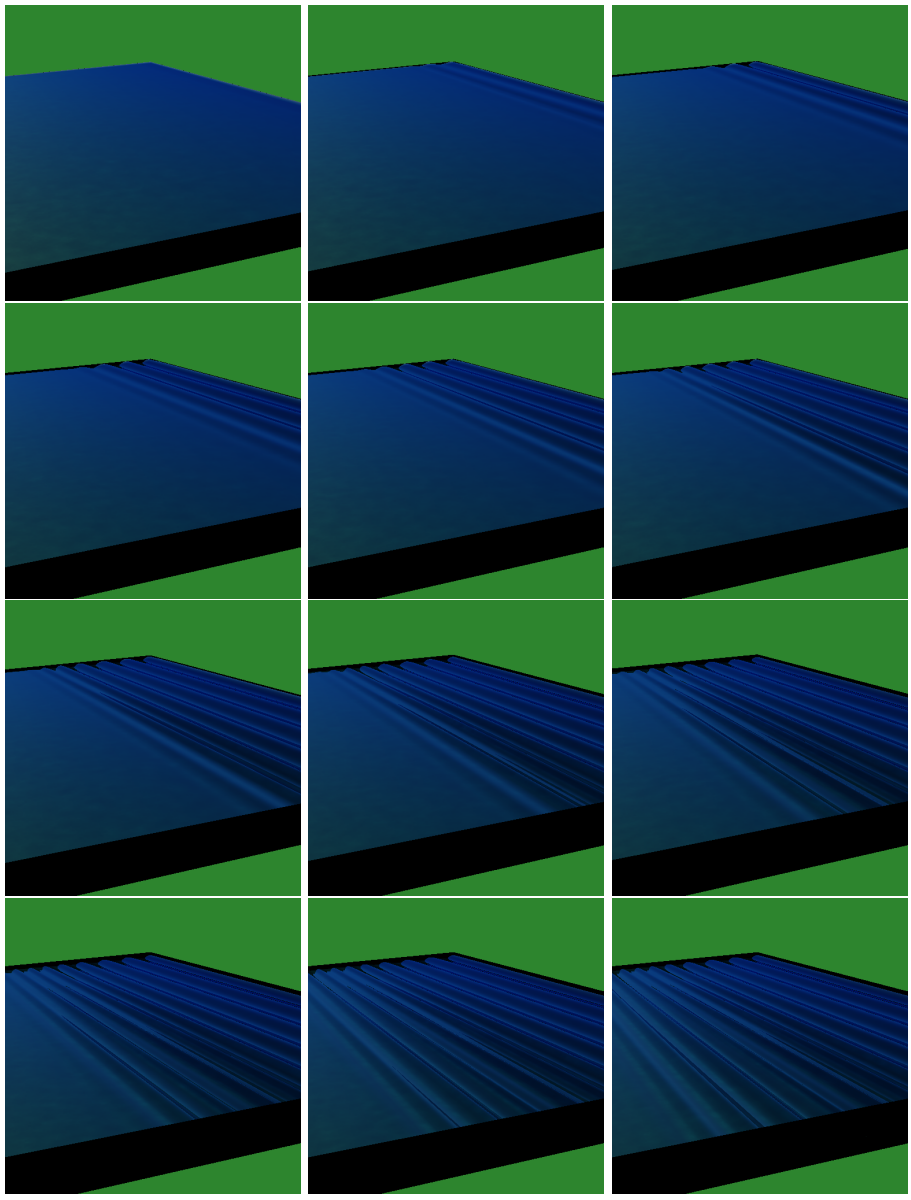

Figura A.2: Esta figura incluye algunos fotogramas pertenecientes a la secuencia de la ola de Shalin descrita en el capítulo 4

Apéndice B

Versión de la memoria en inglés

En este anexo se incluye la versión en inglés de la memoria, que ha sido entregada en la Universidad Técnica de Dinamarca.

Rendering Ocean Wave Simulations

Javier Delgado Aylagas

Kongens Lyngby 2013
IMM-B.Sc-2013

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-B.Sc-2013

Summary

The goal of this thesis is to build a rendering framework for the Ocean Wave simulator developed at DTU Compute [EKBL09]. In addition, the goal of this thesis also covers the rendering of the water meshes generated by the mentioned simulator using real water properties having realistic water as a result.

This project contais two separated parts. The first one covers the conversion of the files generated from the simulator to Wavefront OBJ files which can be used using different modelling tools.

The second part deals with the rendering part, which cover different rendering techniques that have been added together in order to obtain a realistic result. The technique is based in raytracing, altough more techniques have been used to obtain a more realistic and accurate result.

Preface

This thesis was prepared at the DTU Compute department at the Technical University of Denmark in fulfilment of the requirements for acquiring an B.Sc. at the University of Zaragoza in the context of an Erasmus exchange.

The thesis implements a framework to the Ocean Wave simulator developed by Allan P. Engsig-Karup [EKBL09] at DTU Compute Department. This simulator has two versions, one for CPU and one for GPU. This project has been developed using the first one, which has been developed in Fortran.

The Ocean Wave simulator is a tool that allows the generation of water meshes using a high variety of properties. The output of this simulator is a binary file containing all the information to generate a 3D mesh.

The thesis consists of different methods which generate Wavefront OBJ files and a visualization and rendering framework in which different shaders will be implemented using real water properties. In this project, different shaders and rendering techniques are going to be combined in order to obtain realistic water.

The application under this thesis has been set up to import correctly every possible mesh generated by the simulator and it can be improved using newer or more complex techniques allowing future developments.

Lyngby, 05-Septiembre-2013-2013

Javier Delgado Aylagas

Acknowledgements

I would like to thank my supervisor Jeppe Revall Frisvad for his support during all the development of this thesis, but also for providing the rendering framework, which has considerably simplified the technical implementation of the prototype.

I would also like to thank Allan P. Engsig-Karup for providing the ocean wave simulator, which is one of the most important parts of this project. And I would like to thank Stefan Lemvig Glimberg and also Allan P. Engsig-Karup their support with the simulator execution.

I would also like to thank DTU-Compute for hosting this project and specically J. Andreas Bærentzen for providing, together with Jeppe Revall Frisvad, weekly feedback throughout the entire working process.

I thank you also the company Force Technology their interest in this project and also their involvement in it providing some of their simulations.

Finally, I would like to thank both the Technical University of Denmark and the University of Zaragoza for allowing me to stay in Denmark where I have developed this thesis.

Contents

List of Figures

Chapter 1

Introduction

Computer generated water is a very used element nowadays. It is being used in a lot of applications but it is mainly used in video generation for films or adverts, and for computer games, but also a lot of companies need water rendering for investigation and also for simulations.

Water can be very difficult to render, and the proccess can be separated in two steps. The first one is the water geometry which must be updated every frame if we don not want completely calm water. The second step is the rendering of the geometry and it will handle with the water properties as a material.

In this project, the first step is performed by a simulator developed by A. P. Engsig-Karup, Morten G. Madsen and Stefan L. Glimberg at the Department of Informatics and Mathematical Modeling [EKBL09][EKMG12].

The aim of this project is to provide a rendering framework which takes as input the ocean wave simulations generated by the mentioned simulator. This framework will have realistic water appearance as output in a process that requires two steps. First of all, it has to deal with the compatibily between the simulator and the rendering framework, and the second step deals with the techniques used to obtain realistic water.

This project will study the most important water properties which are going

to be used to obtain realistic water. Some of them are the seafloor colour and its distance to the water surface, but also the sky and environment which also affect its aspect as they are reflected by the water.

The render engine is based in raytracing and it is completed with photon mapping as water is known to generate caustics in the seafloor. The render engine will be also set up in order to show the generated correctly, and it will also allow the user to generate sequences of frames as well as the simulator does, so ocean water meshes can be generated massively to produce video sequences.

This project uses a visualization framework used in the course Phisically Based Rendering to implement different techniques. In this project, only some of them have been implemented but it has been extended in other many ways.

1.1 Expected outcomes

The project has got two separate parts. The first one deals with the simulator. In this part, the main parameters of the simulator input file will be explained. In addition, this part covers the transformation from a binary file generated with the simulator and its conversion to a Wavefront OBJ file which is the input of the render engine. This step is performed using Matlab.

The second part covers the adjustments made to the render engine but also the shading step, which is the main purpose of this project. On the one hand, the framework has been modified and completed in order to get the most accurate results. Also, it may add missing meshes such as the seafloor in the case that it is not provided by the simulator. On the other hand, different shaders have been used in the proccess adding complexity starting from a simple transparent shader and completing it until the final one.

Finally, the outcome of the project as a whole, is a variable number of pictures, which can be combined to generate video files using third party applications.

An example of the output image file can be seen in the figure 1.1.

1.2 Document structure

The content of the rest of the document is organized as follows:

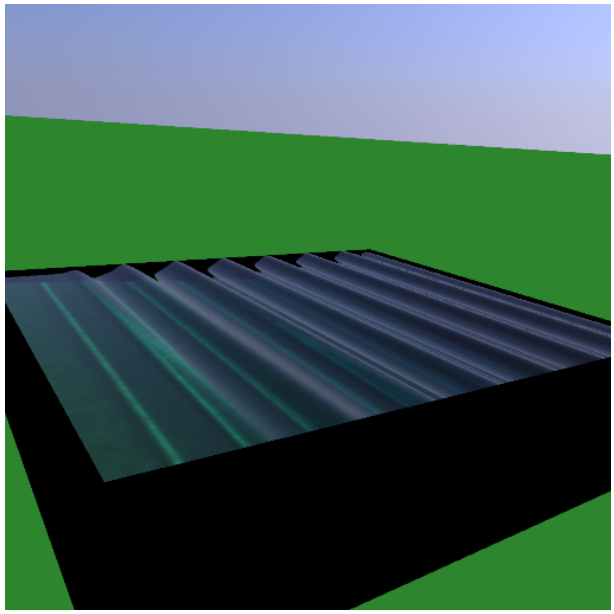

Figure 1.1: Example of the result of the execution. The seafloor can be appreciated, and also the depth of the water and the caustics generated by the waves.

Rendering. This chapter contains all the information related with the simulator and the rendering framework. It explains the main parameters used for the water, but also how are the meshes converted into Wavefront OBJ files and what has been changed in the framework to open the simulator files correclty.

Shading. This chapter explains in detail the different implemented shaders and the techniques used in all of them.

Results. This section analyzes the execution time of the render and also shows the aspect of the different simulations both in Matlab and after the rendering step.

Conclusions. This chapter details the conclusions of the project, but also its limitations and future improvements to continue its development.

Chapter 2

The rendering pipeline

Water surfaces are very common in video games and films, and it is usually a critical element and its level of detail will improve the realism of any scene. In addition, it is usually a very hard computational problem so that it is still difficult to render real-time water [JL04][Kry05]. For that reason, this project will try to compute realistic water with short rendering time leaving real time rendering to the future.

This chapter explains how has been the work organized. First of all, the simulator generates water meshes exported as binary files. Afterwards, these binary files should be transformed into Wavefront OBJ files. This step is performed using Matlab functions. Finally, the object files are imported into the rendering framework which, used as it is explained in the next chapter, will output PNG image files.

The complete pipeline that is covered by this project is shown in the figure 2.1.

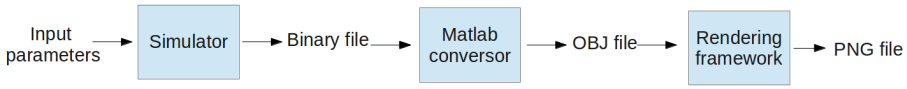

Figure 2.1: Pipeline of this project

2.1 The simulator

This section will situate the context of the project, which aim is to create a rendering framework for the Ocean Wave simulator developed by A. P. Engsig-Karup, Morten G. Madsen and Stefan L. Glimberg at the Department of Informatics and Mathematical Modeling [EKBL09][EKMG12].

The simulator has been developed using FORTRAN and it runs under UNIX machines. The simulator has got CPU and GPU versions. In this project, the used version has been the CPU one.

When using the simulator, a lot of parameters can be used to perform different kinds of waves. Some of them have to do with the timing between steps. As the desired frequency to obtain a good quality video is 25 frames per second, these parameters should be adjusted properly. Also, we should use the maximum time value which will determine the length of the video. The parameters used for this project are only examples but the rendering framework will handle every possible output.

2.1.1 Converting the output to a Wavefront OBJ file

The output of the simulator has got two different formats and they can be chosen in the simulator input file. The mesh can be exported as a text file or a as a binary file. In this project the binary format has been chosen. It contains the vertex coordinates but also its energy, which is required if the user wants to continue with the simulation from that poing, but it is not used in this project.

The next immediate step is to convert these files into wavefront object files to make them readable by the rendering framework. This step is performed using Matlab.

The Matlab conversion function loads all the files in the current directory which follow the format “EP_xxxxx.bin” which is the simulator default naming and

converts them into Wavefront OBJ files. Among other things, it also adds some lines to group the objects but it also sets the material file used for the geometry. The default material file is "flow.mtl". This file contains the specular, diffuse and ambient values of the material, but also also the illumination value. The illumination value is the parameter that will determine the shader used in the framework. This value will be changed in the context of this project to show different shaders.

The chosen function deals with many input files from the beginning as it is required for this project. A PNG output from the Matlab function is not neccessary but it has been used to check correctness before going further with the next steps in the pipeline. The mentioned image file looks as it is shown in the figure 2.2.

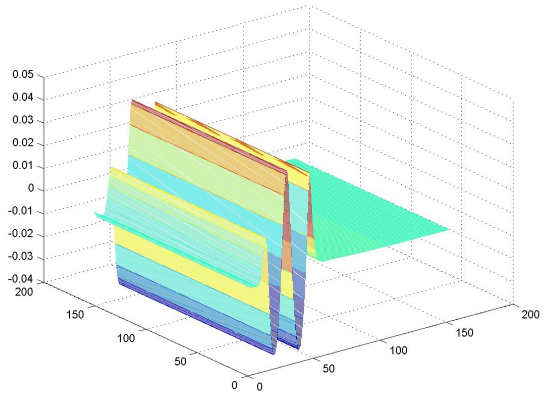

Figure 2.2: This the aspect of the mesh visualized inside Matlab

2.2 The rendering framework

This project uses a rendering framework based on raytracing [App68]. As the time to build a framework from scratch would take a lot of time, the raytracer has been provided to use in this project. It has been provided by Jeppe Revall Frisvad and it is used in the course Physically based rendering at the DTU Compute Department. It contais the main properties of a raytracer in which the different shaders will be implemented. It allows to import Wavefront OBJ files and it uses a BSP (Binary Space Partition) tree [SS92] to store the geometry

in memory.

Inside the rendering framework, the user is allowed to move the camera using the mouse, save and load camera positions and views, save images as bitmaps or increment or decrement the number of rays per pixels to increase accuracy and reduce aliasing, among others.

The framework also handles different views that show different effects such as refelectance, direct lighting only, ambient occlusion, path tracing, photong map caustics and others. In this project only path tracing and photon map caustics will be used. The essential properties of these techniques are included in the framework altough they were not implemented. For this reason, first of all it is needed to check which techniques fit better in the context of the project and only those will be implemented. All the techniques mentioned outside this section have been implemented during the project development.

2.2.1 Adaptation of the framework to the simulator input

The first versions of the simulator do not generate a seafloor, which is required by the definition of this project. In this case, a plain rectangle will be located under the geometry to act as a seafloor. As this option is not accurate because the seafloor affects the motion of the waves, the last versions of the simulator generate automatically a correct seafloor.

In addition, as the light can reach the geometry from every point, it could reach the seafloor without going through the water as it is shown in the figure 2.3.

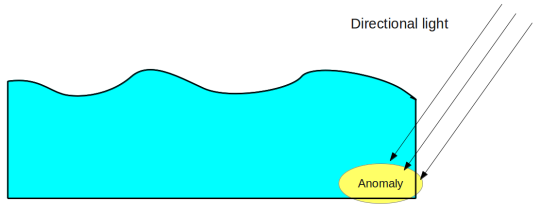

Figure 2.3: This diagram shows why a bounding box is needed. If there is no bounding box, the light can reach the seafloor without going through the water. With a bounding box there is still an anomaly, because the corners may be in shadow.

To solve this issue, a bounding box has to be placed in the four sides of the

water mesh, and it must cover, at least, the height from the seafloor to the water surface. A plane has also been included as a floor of the visualization environment.

By using a bounding box, there is an undesired effect, which is that depending in the angle of the sun, there will be a small shadow inside the water caused by the walls of the box. These shadows may affect more than one wall depending on the inclination angle of the sun Moreover, the opposite wall will also accumulate photons as it is exposed directly to the sun light. This issue can be improved by making bigger meshes or having a smooth transition of the seafloor avoiding the use of walls.

The meshes are not centered in the ground plane because this helps the viewer to visualize the environment as it is easier to see part of the sky. This also creates a shadow in the ground plane. This issues can be avoided with infinite or very big water meshes.

2.2.2 Video production

The video production is handled using command line arguments. In these arguments the user has to define the number of the first and the last epoch to proccess but also the space between epochs which is defined in the simulator input. The file names are handled automatically as the binary files are always named as "EP_xxxxx.bin" and the object files are named "EP_xxxxx.bin.obj".

As long as the number of the last epoch is higher than the first one, the framework will produce a sequence of files which are the frames for the video. The naming of the images follow the same pattern and they will be named "EP_xxxxx.bin.obj.png". The video has to be mounted outside the framework using all the generated files.

Chapter 3

Shading

This chapter covers the main properties that affect the aspect of water. It explains why are they important and which are techniques used to deal with those properties and it also explains the difficulties encountered. It explains the steps followed to build a complete shader for the project.

The first shaders have built separatedly while the final one uses most of the properties of the previous ones. In addition, other complementary properties have been used such as the sun and sky colour [Lew93][Kaj86].

The framework can handle any number of lights, although in this project only one directional light which will represent the sun.

In addition, some of the implemented shaders use a path tracing technique. This is the case of the transparent shader among others. [PH04]

As it was explained in the previous section, the main properties of the materials are defined in a material file (extension mtl). That file contains more than one material and all of them can be used in this project, but it is also possible to define new materials. Complementary to the mentioned material file, there is another file called "media.mpml", which has been also provided with the framework. This file contains more information about some materials. One of those materials is seawater and we are going to use it in this project. As water colour

is different depending from one ocean to another, more kinds of water could also be defined in this file.

3.1 Lambertian reflectance

This shader is the most basic one that has been used in the project, but it is neccessary as it is used for the seafloor.

Lambertian reflectance is the property that defines a pure diffuse surface. The amount of light returned by the geometry depends only on the angle between the light respect the geometry normal and it does not depend on the eye position. The Lambertian BRDF (Bidirectional reflectance distribution function) can be seen in figure 3.1. As an example, the scene has been rendered using this shader for the water. The result can be seen in figure 3.2.

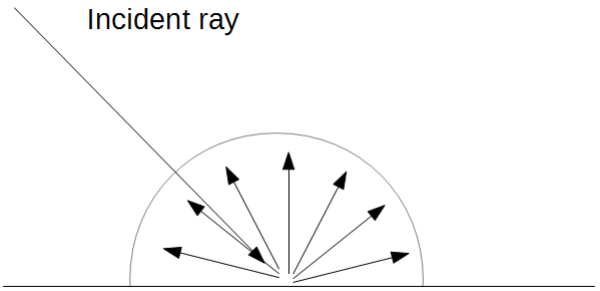

Figure 3.1: Lambertian BRDF

3.2 Transparent shader

In order to use this shader correctly, the fresnel equations have been used to calculate the refractive index. The fresnel equations calculate the reflectance, which is the amount of energy reflected while The rest of the energy (1-R) is refracted. Using this equations, the reflectance will vary depending on the incident angle and the index of refraction of both mediums and for small angles, there might be only reflected light [Ska06].

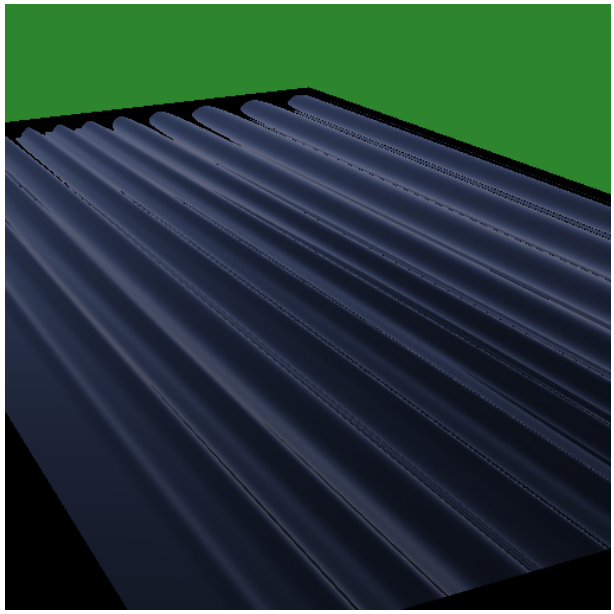

Figure 3.2: Water rendered as Lambertian

Once the reflectance is calculated, this shader traces 2 rays: the reflected one and the refracted one, and they are combined depending on the refractive index that has been explained before [JB02]. The diagram of the reflected and refracted rays can be appreciated in figure 3.3. In addition, the framework uses a variable which sets the maximum number of recursions of the algorithm [PH04].

3.3 Photon mapping

In this project, we are using a seafloor which will affect the aspect of the water in different ways. One of that ways will be the caustics produced by the waves which will be seen in the seafloor.

As caustics are going to affect the aspect of water significantly, photon mapping has been implemented in this project [NJC00]. The photons may produce caustics depending on the shape of the wave but also depending on the distance from the seafloor to the sea surface.

As we have explained before, the framework allows the use of photon mapping

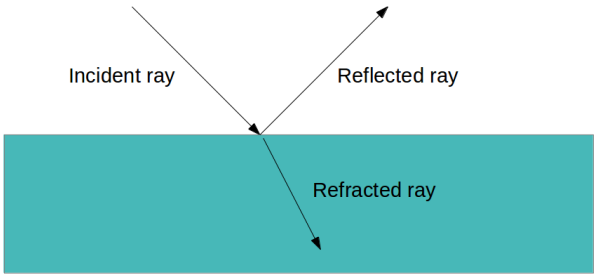

Figure 3.3: This diagram shows the reflected and refracted rays, which are used in some of the shaders.

altought it must be implemented. In addition, there is an option to visualize the photon maps. These photon maps and the whole scene using a transparent shader with caustics are shown in the figure 3.4.

The number of used photons can be set in the framework, and also the number of photons used in the estimation. In this project, these values have been set to 7500000 photons and 200 of them used for the estimate.

3.4 Absorption

The next effect that we are going to use has to do with the depth of the water. The darkness of the water will increase with its depth. This phenomenon is called absorption.

This shader was projected to be a volume shader, but as the meshes returned by the simulator are not volumes, this idea is not applicable. Instead, this shader calculates the distance from the water surface to the seafloor in order to calculate the quantity of energy absorbed. The distance is calculated using the direction of the refracted ray from the water surface so it will be longer or equal than the perpendicular distance from the water surface to the seafloor. [EC05]

Figure 3.5 shows the scene using absorption but no photon mapping in this case.

At this point the reader has to realize that the used shader is not the transparent

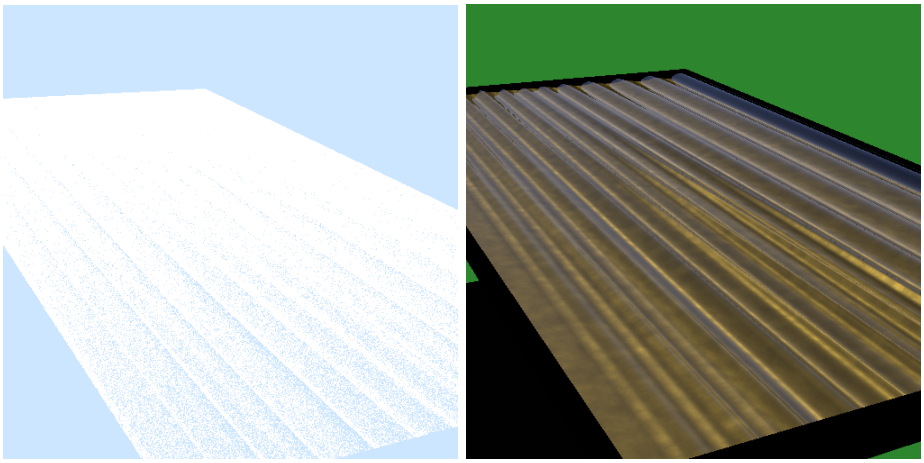

Figure 3.4: Left: Visualization of the photon maps. Right: Render of the scene using a transparent shader with photon mapping. The caustics can be appreciated at the seafloor

one anymore. Now is where the bounding box makes sense, because the only way in which can be light is inside the water is through the surface. In other words, the light that reach the sea bottom is because of the photons that have crossed the water. Afterwards, the light inside the water may not reach the surface again due to absorption, which will determine the final aspect of the water.

3.5 Phong reflection model

In order to include another property to the simulator, the phong reflection model has been used to reflect the sun. This model is not the Phong shading model. Its contibution is only the reflection of the directional light and it will be appreciated only if the eye, the water surface and the sun are situated in the same plane [Pho75].

Figure 3.6 shows the scene using with phong reflection added to the rest of properties.

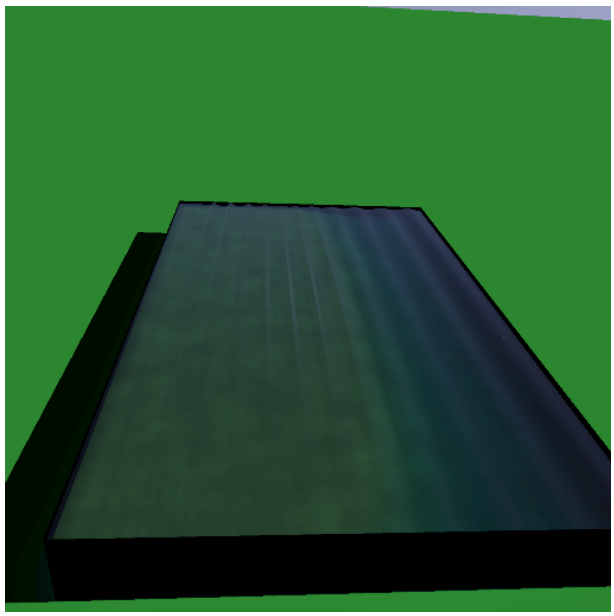

Figure 3.5: This figure shows how the colour of water is affected by absorption. In the left side the depth is lower and the result colour is lighter because of the seawater colour. In the right side of the scene, the colout is darker because the depth is higher.

3.6 Other properties

This section explains some other minor properties that have been used along the project.

Sun and sky

The sky is also important as it is, either completely, or almost part of it, reflected by the water. The chosen model for the day light is the one developed by A. J. Preetham, Peter Shirley and Brian Smits at the University of Utah [PSS99]. This model uses real coordinates of the Earth but also the desired date and time.

For this project, the chosen date is a day in autumn at 12.00 and it has been located in Denmark. These values can be changed at any time in the framework.

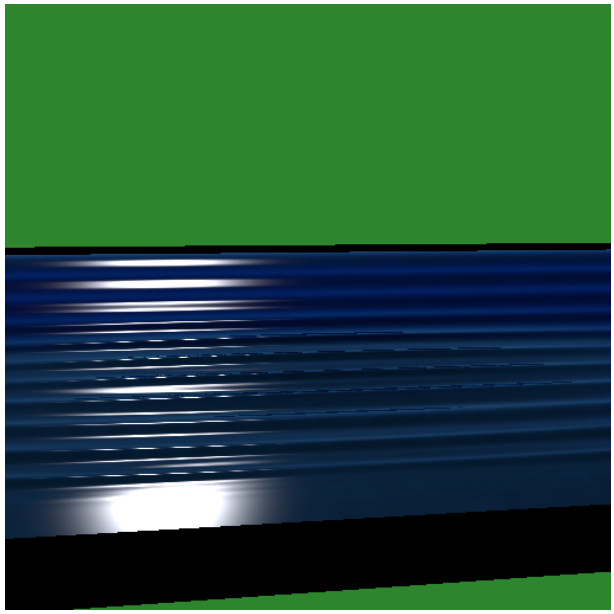

Figure 3.6: This figure shows how is the sun reflected in the water surface because of phong reflection.

Antialiasing

In order to make the result more accurate and avoid effects as aliasing, most of the generated images have been rendered using multisampling. The framework allows the generation of more than one ray per pixel and it has been used in this project. However, as the rendering time increases very fast, the maximum number of rays per pixel used is 9. For video generation, as it is needed to render a high quantity of frames, only one ray per pixel has been used.

3.7 Final comments

As it has been explained before, in the final version the user can choose between different shaders. This is handled in the file "flow.mtl" and the material used for the water is "seawater". Inside this file, there is a value called "illum" which determines the shader that is going to be used in the renders. By default, this value is set to "15", which is the shader used for ocean water. This shader uses absorption, photon mapping and phong reflecion. This shader has been used,

for example, in figure 3.7.

Other used shader is the transparent one, which can be chosen changing the illumination value to "4". This shader uses a basic transparent shader with photon mapping. This shader has been used in the right image in figure 3.4.

Finally, the lambertian shader has been used for the ocean water in figure 3.2 and it has been used for the seafloor in all the renders.

In addition, the sun and sky model has been used in all the renders as it is not affected by any of the shaders.

The whole pipeline that the user must follow can be checked in chapter 2

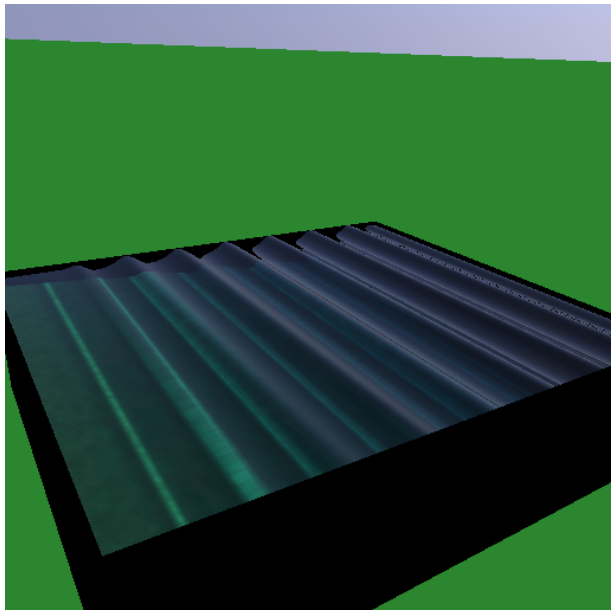

Figure 3.7: This figure shows the caustics and absorption.

Chapter 4

Results

Once the implementation has been completed, three simulations have been run in order to study the time consumption. Two of the results shown here come from renders that have been configured to be a video sequence. The other one has been performed for a single image.

This has been done because usually the mesh is plane in the first frame, and as the time increases, the variations in the meshes are higher and it affects to the render time. The time frequency for all the simulations is 25 frames per second, which is the standard for european televisions. This means that the step time is 0.04 seconds.

All the simulations and renders have been performed in my personal laptop. The render times will be lower using a more powerful computer. In addition, in the cases of video sequences, the obtained times have been performed using only one ray per pixel. For more rays per pixel than one, the time is approximately multiplied by the number of rays per pixel. In the case that the render is focused in one single image, the number of rays have been set to 9 in order to get more accurate and nicer images.

4.1 Linear travelling Wave

This simulation uses only linear techniques to obtain the simulations, so it is expected to be computationally easy. The simulation will generate a sequence of 600 meshes and it will represent a 24 seconds video with a frequency of 25 frames per second. Figure 4.1 shows the time of all the performed steps.

LINEAR	600 frames	Average
Simulation	3 minutes	0,3 seconds
Conversion	42 seconds	0,07 seconds
Rendering	83,3 hours	8,3 minutes

Figure 4.1: This table shows the time for the Linear Travelling Wave example

In this case, the simulation has taken 0.3 seconds per frame so the total time has been 3 minutes for the whole sequence. The conversion time is also significant, but this step is faster than the others. In this case, the conversion time has taken an average of 0.07 seconds per frame, making a total of 42 seconds for all the images.

The mesh visualized in Matlab and the render result can be seen in figure 4.2 although the outputs of this simulation have been also used in the previous chapters of this document.

Finally, the rendering step has taken 8.3 minutes per frame, making a total of 83 hours for the whole sequence. The size of the meshes used in this example is 259 x 2.

4.2 Whalin's experiment

Robert W. Whalin, Ph.D., P.E. is Associate Dean and Professor of Civil Engineering College of Science, Engineering, and Technology, Jackson State University. This simulation uses some of the data gathered in the Whalin's experiment [1].

[1] http://coastalhazardscenter.org/people/robert-w-whalin

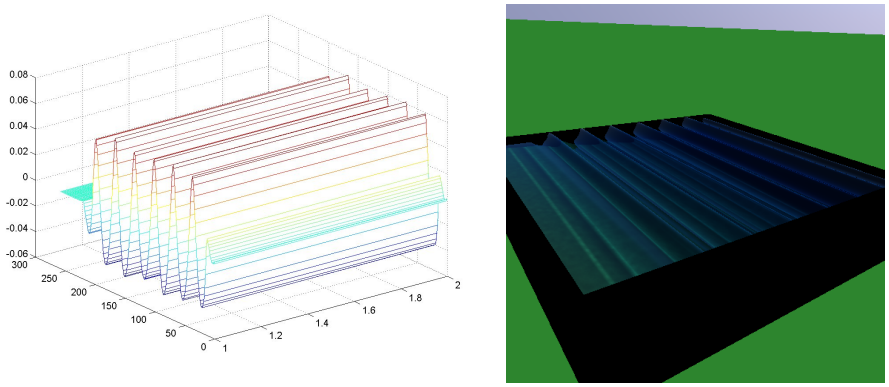

Figure 4.2: Visualization of the Linear Travelling Wave inside Matlab and after the rendering

This simulation will generate again a sequence of 600 meshes and it will represent a 24 seconds video with a frequency of 25 frames per second. Figure 4.3 shows the time of all the performed steps and also the time at different points of the simulation. In this case, the size of the mesh is also bigger than in the previous one.

WHALIN	600 frames	Average
Simulation	110 minutes	11 seconds
Conversion	5,5 minutes	0,55 seconds
Rendering	35,6 hours	3,5 minutes

Figure 4.3: This table shows the time for the Whalin Wave example

In this case, the conversion time has been higher than the previous simulation as the water mesh is bigger, but this step has been again the easiest to compute.

Finally, the render time has been 3.5 minutes per frame, making a total of 35 hours for the total of 600 frames. Figure 4.4 shows the mesh visualized inside Matlab and also the final render. The mesh size in this example is 259 x 19.

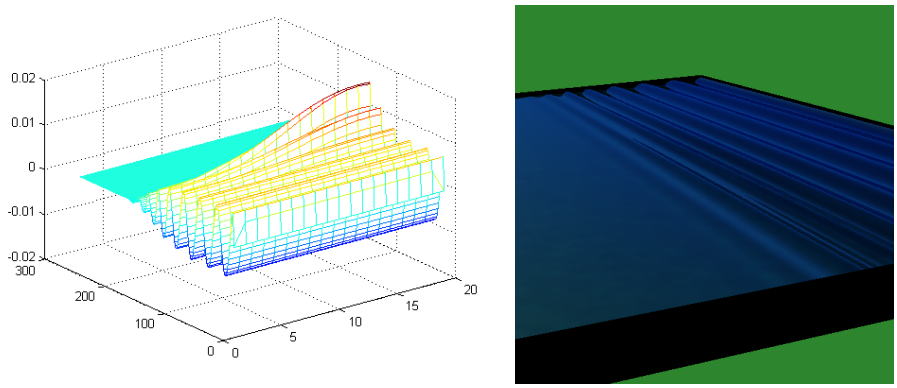

Figure 4.4: Visualization for the Whalin Wave inside Matlab and after rendering

4.3 Newmann Kelvin

The next simulation has been courtesy of Force Technology. It includes a water surface as well as a ship hull. The ship has no deck but it is not needed for the example.

The provided meshes have been gerenated using the same simulator and also a ship model from the company. The provided information aspect in Matlab is shown in figure 4.5.

This simulation uses a format quite different to the previously used. With the meshes files there has also been included a Matlab file to read them, but they have to be adjusted to be coherent with the rendering framework input. In that way the previous conversion function has been redefined for this precise example. The new output includes two meshes inside one file, the first one is the water mesh and the second one is the ship hull.

As this simulation represents only one frame, it has been rendered using 9 rays per pixel. The file conversion and rendering time can be checked in figure 4.5.

The resulting render is figure 4.7

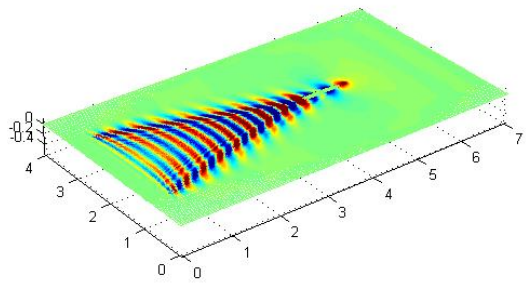

Figure 4.5: Aspect of the simulation provided by Force Technology as it is seen inside Matlab

NEWMANN	1 frame – 9 rays per pixel	Average (ray)
Simulation	-	-
Conversion	16 seconds	-
Rendering	53 minutes	5,8 minutes

Figure 4.6: Time of the Simulation provided by Force Technology

4.4 Comments

For every simulation, some adjustements have been needed in the framework to get better images. These changes have been done to all the renders.

One modification is that the Z coordinate of the meshes has been scaled because in the original mesh, the variations of Z were hardly visible. Moreover, there is a parameter that affects the absorption and caustics terms which scales the distances but not the mesh. This means that, altough the mesh is the same, it will represent deeper or less deeper water.

As a conclusion for this chapter, the simulations have been as accurate and realistic, but not as faster as expected. One possible way to make simulations faster is to carefully adjust the mesh size so that it is still accurate but without

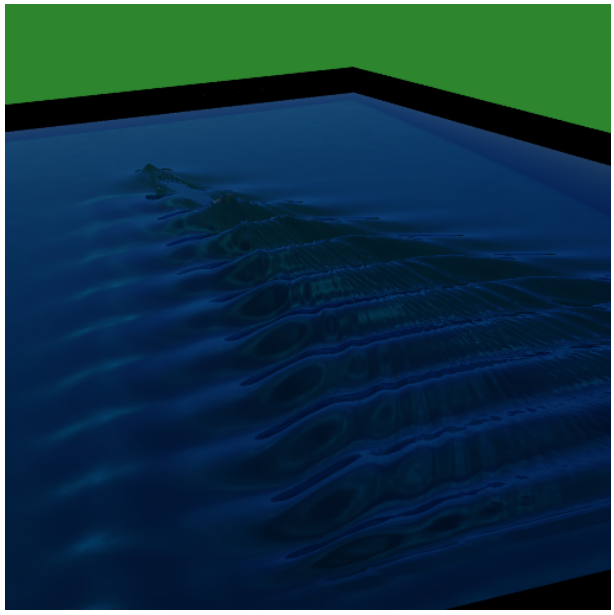

Figure 4.7: Render of the simulation provided by Force Technology

unneded polygons.

Chapter 5

Conclusions

This project has developed a complete framework to use combined with the Ocean Wave Simulator [EKBL09] starting from a raytracer. This framework combines many different techniques that allow the user to render realistic water using different techniques that have been combined together and it also provides simple shaders so that external objects can be added to the framework using the material file. In the one hand, the developed render framework is a very powerful application because it could also be used with any other mesh and it will render it as water because all the needed objects are added inside. In the other hand, although the result is accurate and realistic although it has been slower than expected.

This project has also developed a Matlab conversor which produce OBJ files that can be used also in any other 3D software so its capabilities go beyond the aim of this project. In addition, there is an extra Matlab file that has been used to export the files provided by Force Technology and which has been used in chapter 4.

5.1 Limitations and future improvements

This section will explain some of the known limitations of the rendering framework and some future improvements.

The main limitation of the framework is that the camera cannot be moved during the rendering. It can be moved anywhere before starting rendering so it works good for pictures, but for videos the camera is in the same place until the video has finished its rendering.

The simulator main function is to generate open sea meshes. This means that no environment is handled in the simulator. As the environment affects to the water aspect significantly it would be a good option to include it in future improvements. But, even in the case that a environment is added to the rendering enviroment, it would not be physically coherent because the water meshes would not interactuate with the terrain. The only way to be totally physically coherent would be to include the environment in the simulator and export the whole scene in the binary file.

As it has been commented in previous sections, some shadows are caused by the bounding box. Altough this solution is better than no having anything, the aspect is not so accurate near the bounds.

Moreover, in this project it is very difficult to programme a sequence of renders. For example, if the user wants different renders of different meshes or the same one from different points of view, it has to be set up manually and generate them one by one. Although the framework has functions to save and load camera coordinates or object files, it should be done inside the code.

Other limitation is that the simulator has an output of one or two meshes in one file. The first mesh is, of course, the water surface mesh and the second one may be the seafloor in the case that it exists. However, usually these two meshes do not intersect each other. If they intersected, for example finishing in a beach, the problem of the shadow near the walls could be solver automatically. In addition, if the simulator allowed to import object files into it, it would generate coherent waves intersecting with, for example, terrain, solving one issue that has been explained earlier in this chapter.

Finally, as it has been commented previously, the rendering step has been slower than expected. The simulations and renderings have run over my personal laptop, and altough more powerful machines could be used, future developments should go further. As the simulator has a newer version that runs on GPU, that version could be used instead of the CPU one. In addition, following the same

idea, some shaders could be moved from the CPU to the GPU too. This would be the case of reflection and refracion.

5.2 Personal conclusions

With this project, I have learned a lot of how projects are developed in the real world, how are they organized and how are they scheduled.

This project has been also a challenge because I had to work with a very big framework which I have had to learn before starting to improve it. Thanks to it, I have also improved my knowledge on computer graphics and rendering.

In addition, having developed this project during my exchange stay in Denmark has shown me how the departments outside the University of Zaragoza work.

5.3 Project development

The project has had a duration of seven months. The first five months have had the hardest work and there have been weekly meetings with the graphics group of the DTU Compute department. Also some private meetings have been neccesary in order to get the project in the correct way.

There had also been an extra meeting with the company Force Technology located in Kongens Lyngby, Denmark. The dealt topics were the suitability of the simulator and the framework to be used in the boat simulators of the company and the availability of use some of their boat models. Finally, there were no further meetings as this project was not meant to be real time water rendering altough they provided one of their own simulations combined with a ship which have been used in chapter 4.

This is the Gannt diagram that describes the evolution of this project.

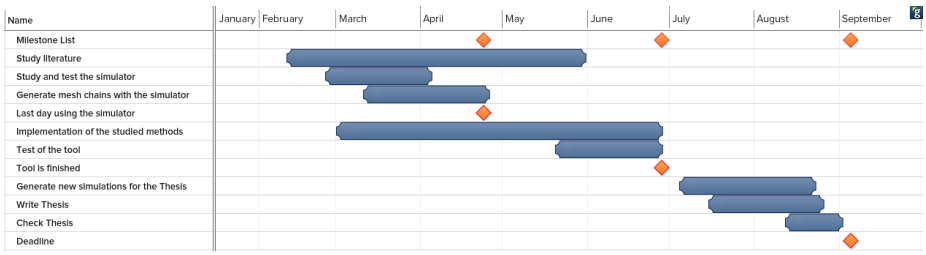

Figure 5.1: Gannt diagram

Bibliography

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[EC05] Xavier Pueyo Francisco J. Seron François X. Sillion Eva Cerezo, Frederic Pérez. A survey on participating media rendering techniques. 2005.

[EKBL09] A. P. Engsig-Karup, H. B. Bingham, and O. Lindberg. An efficient flexible-order model for 3d nonlinear water waves. *J. Comput. Phys.*, 228(6):2100–2118, April 2009.

[EKMG12] A. P. Engsig-Karup, Morten G. Madsen, and Stefan L. Glimberg. A massively parallel gpu-accelerated model for analysis of fully nonlinear free surface waves. *International Journal for Numerical Methods in Fluids*, 70(1):20–36, 2012.

[JB02] Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph.*, 21(3):576–581, July 2002.

[JL04] Claes Johanson and Calle Lejdfors. Real-time water rendering. *Lund University*, 2004.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[Kry05] Yuri Kryachko. *Using vertex texture displacement for realistic water rendering*, volume 2. 2005.

[Lew93] Robert R. Lewis. Making shaders more physically plausible. Technical report, Vancouver, BC, Canada, Canada, 1993.

[NJC00] Henrik Wann Jensen Niels Jørgen Christensen. A practical guide to global illumination using photon maps. 2000.

[PH04] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[PSS99] A. J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 91–100, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[Ska06] Johannes Skaar. Fresnel equations and the refractive index of active media. *Phys. Rev. E*, 73:026605, Feb 2006.

[SS92] Kelvin Sung and Peter Shirley. Graphics gems iii. chapter Ray tracing with the BSP tree, pages 271–274. Academic Press Professional, Inc., San Diego, CA, USA, 1992.