UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS

UCI

FACULTAD 5 REALIDAD VIRTUAL

Visualización de Superficies de Lagos Y Océanos

TRABAJO DE DIPLOMA EN OPCION AL TITULO DE INGENIERO EN CIENCIAS INFORMATICAS

AUTOR

Marlon Hernández Magdariaga

TUTORES

Ing. Yaíma Nodarse Valdés

Ing. Lien Muguercia Torres

Ciudad de la Habana

Junio 2008

"El único autógrafo digno de un hombre es el que deja escrito con sus obras."

José Martí

Declaración de Auditoria

Declaro que soy el único autor de este trabajo y autorizo a la Facultad #5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Autor:

Marlon Hernández Magdariaga

_____

Tutor:

Ing. Lien Muguercia Torres

_____

Ing. Yaíma Nodarse Valdés

_____

Datos de Contacto

Autor

Nombre: Marlon Hernández Magdariaga

Correo Electrónico: mmagdariaga@estudiantes.uci.cu

Tutores

Nombre: Lien Muguercia Torres

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Informática

Categoría Docente: Profesor Instructor

Correo Electrónico: lmuguercia@uci.cu

Nombre: Yaíma Nodarse Valdés.

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Informática

Categoría Docente: Profesor Instructor

Correo Electrónico: ynodarse@uci.cu

Agradecimientos

A todas las personas que me han apoyado en la realización de este proyecto, a mis dos tutoras Lien y Yaíma, al compañero Fidel ,a las Fuerzas Armadas Revolucionarias y en especial a la Universidad de las Ciencias Informáticas por realizar mi sueño.

Dedicatoria

*Este trabajo de diploma se lo dedico a toda mi familia en especial a mi Mamá y Papá que me alentaron todo este tiempo y aguantaron todas mi malacrianzas y me supieron guiar por el camino que tome, a esas personas que han permanecido conmigo en los momentos buenos y malos, el Niche, el Leo, el Rica, el Kandiman, a Helmo. Se lo dedico a ese lugar donde ahogue mis penillas de vez en cuando, Shanghái y también a los que me ayudaron en especial a Aly**.** A Yani Lis, que no me puso en sus agradecimientos mira esto es para qué veas que no te guardo rencor y en especial a Consuelo, Anita que me apoyaron hasta el final y cuando estuve enfermo, a mi novia Yasmina que sin ella realmente no hubiera terminado y a mí que no se me ha subido la fama para la cabeza y fui el que le metió coco a esto.*

Marle.

Resumen

Lograr una visualización realista de la superficie de Lagos y Océanos es un tema polémico de discusión en el mundo de realidad virtual y especialmente en los gráficos por computadoras. Alcanzar un nivel de realismo deseado requiere de elevado consumo de recursos y una alta complejidad en cuanto los algoritmos lograr dicha representación y a veces no se cumplen con las expectativas deseadas. Este trabajo propone una solución para proveer de un mayor realismo a las escenas virtuales a través de la reflexión y refracción del entorno, y la deformación de la superficie.

Partiendo del estudio de herramientas y algoritmos que den solución o respuestas al problema de la visualización realista de la superficie de Lagos y Océanos, se nos deslumbran características que aparean ventajas y desventajas .Se propone desarrollar un módulo que visualice realistamente la superficies lagos para su aplicación en entornos virtuales en 3D a través de los shaders (tipo de lenguaje de programación destinado a programar el procesado de elementos de cualquier interfaz 3D o tarjeta graficas), y además se propone algoritmos para la deformación de la superficie de agua que asemejen las perturbaciones que ocurren en la naturaleza donde vivimos en correspondencia con las necesidades de los clientes.

Como resultado, este módulo se podrá acoplar a las herramientas de desarrollo de aplicaciones de Realidad Virtual en la Universidad de Ciencias Informáticas, aportándole una tecnología de primera línea con un alto nivel para herramientas de este tipo de sistemas a escala mundial.

Índice

Introducción

El desarrollo de la informática ha permitido la popularización del término:"Realidad Virtual". De los programas militares de entrenamiento, simuladores de vuelo, entre otros, ha saltado a los programas domésticos y profesionales. Hoy en día, en Cuba, se empieza a aplicar en la medicina, la arquitectura, la enseñanza y en los programas de ocio, entre otros muchos campos. Pero sus futuras aplicaciones son imprevisibles dada la cantidad de áreas en los que puede ser utilizada.

La realidad virtual (RV) entra en un espacio de la informática que debido al gran avance de las herramientas y tecnología no se puede delimitar fronteras, es decir, no se puede marcar pautas de comienzo ni de fin, su vitalidad está dada por la capacidad de imaginación y potencialidades de ser capaz de generar mundos artificiales en ordenadores que posea el hombre. Logrando hacer de este un sector donde ingenieros, desarrolladores y especialista sobre el tema encuentren un mayor atractivo.

La Universidad de las Ciencias Informáticas desde su inicio se ha incorporado, con un aporte creciente, al desarrollo e investigación del tema que nos acoge hoy "La Realidad Virtual", mediante de la Facultad #5 la cual tiene adjudicado dicho polo.

El Polo de Entornos Virtuales esta subdividido en proyectos productivos, cada uno con un campo distinto en el mercado: desarrollo de juegos, simuladores de conducción, simuladores para la defensa (tiro), simuladores quirúrgicos, etc.

Hasta ahora en los proyectos productivos que incluyen de alguna forma paisajes virtuales se ha hecho representaciones rudimentarias de tipo 2D de pequeños y grandes cuerpos de agua como lagos y océanos respectivamente, las cuales poseen poca calidad visual y poco realismos, pero nada referente ha efectos visuales como la reflexión y refracción del entorno y deformación de la superficie del agua. Esto conlleva a que se limiten a no usar estos efectos visuales en sus aplicaciones, reduciendo su posibilidad de emerger en el mercado mundial debido a la falta de la realidad y calidad del software.

Este módulo serviría como una herramienta de funcionalidad común que podría utilizarse en aquellos proyectos que necesiten representar la superficie de los lagos para un entorno virtual de un producto, además se ayudará con materiales que servirían de apoyo para estudiar la deformación de océanos, y así

se ampliará mas la información acerca de este tema muy importante aplicable también en entornos virtuales, con una finalidad determinada, que impulse al usuario a una mayor compenetración con los simuladores y juegos. Para mantener la ilusión de credibilidad en un mundo de Realidad Virtual se hace necesario una visualización lo más realista posible de lagos, tal que su semejanza con el mundo real sea lo más certera posible, al igual que en la deformación de océanos.

Analizando la situación existente, se propone como **problema científico** a resolver en este trabajo: ¿Cómo visualizar la superficie de lagos y océanos para su aplicación en entornos de realidad virtual?. Para ello se propone la creación de un módulo que reúna los algoritmos que den solución a este problema, de esta manera la representación de la superficie de agua en entornos virtuales constituyen el **objeto de estudio** y la visualización realista de lagos y deformación de océanos como **campo de acción**.

El **Objetivo principal** del trabajo es desarrollar un módulo que visualice realistamente la superficie de lagos y proponer algoritmos para representar océanos para su aplicación en entornos virtuales en 3D para ello se debe lograr que dicho módulo sea completamente transparente al usuario, flexible a actualizaciones futuras, que utilice eficientemente las cualidades de las tarjetas gráficas y que soporte la biblioteca gráfica *OpenGl* .También se realizará el estudio de los algoritmos más usados actualmente ,dirigido a la visualización de lagos y deformación de océanos y se le dará implementación a este módulo.

Se plantean entonces un grupo de **Tareas de investigación** que permitirán satisfacer el objetivo trazado y que se pueden resumir en las siguientes:

- Estudiar las características básicas de los módulos de efectos visuales de un lago.
- Estudiar la deformación de los océanos para sistemas de realidad virtual.
- Analizar algoritmos utilizados en la creación y visualización de los efectos visuales de un lago y la deformación de los océanos para sistemas de realidad virtual.
- Caracterizar tendencias actuales utilizadas en la creación y visualización de los efectos visuales de un lago y deformación de océanos.
- Estudiar las principales herramientas existentes para realizar la simulación de entornos virtuales en 3D en el mundo.
- Describir el funcionamiento del pipeline gráfico y los *shader*.

- Determinar las potencialidades de los lenguajes de alto nivel utilizados en el mundo para desarrollo de efectos visuales de un lago y deformación de océanos.
- Diseñar una biblioteca de clases que de solución a la visualización realista de la superficie de los lagos.
- Proponer Algoritmos para la representación de Océanos de forma realista.

Como resultado de este trabajo se pretende obtener un módulo que sea capaz de visualizar la superficie de Lagos en escenas 3D lo más real posible teniendo en cuenta los efecto de reflexión y refracción y se propondrá un algoritmo para la deformación de océanos. Permitirá a los desarrolladores de entornos virtuales en 3D reutilizar el módulo indistintamente.

Capítulo 2 Fundamentación Teórica

2.1 Estado de Arte

En el mundo actual la representación de Lagos y Océanos es un componente muy usado en entornos virtuales como simuladores o juegos por computadoras. A menudo esta representación no alcanza una perfecta visualización, haciendo que la simulación sea lo menos parecido ópticamente al mundo real. Este problema está determinado por el nivel de realismo que se quiera alcanzar, al desarrollar dicho componente en las aplicaciones que lo utilicen. También esta ineficiencia está vinculada a que lograr un alto nivel de realismo en estos componentes requiere de utilización de algoritmos complejos que de cierta manera afectan el *renderizado* en tiempo real de la aplicación junto a la complejidad inherente de los procesos físicos que participan en la simulación.

Con el avance del pipeline programable, muchos de los algoritmos desechados para el proceso de simulación de lagos y océanos se han puesto disponibles para una visualización eficiente de los mismos. Actualmente corporaciones como NVidia, Intel y ATI, fabricantes de procesadores gráficos (GPUs), presenta productos con la finalidad de mejorar la calidad de visualización de videojuegos y o aplicaciones 3D interactivas [15]. Estos implementan ciertas operaciones gráficas llamadas primitivas optimizadas para el procesamiento gráfico con el objetivo de darle un aspecto más realista a las representaciones de entornos virtuales.

Actualmente la simulación de lagos y océanos no se realiza como la simulación de cualquier objeto en una escena de 3D ya que hay que tener en cuenta sus propiedades para simularlas de una forma realista, por lo general, consta de dos pasos lógicos principales: La simulación de la superficie de agua, que en la actualidad es una dirección de investigación muy popular donde mucho proponen algoritmos basados en las dinámica de fluidos como la resolución de la ecuaciones de Navier-Stoke (NSE) en 2D [3] y 3D [5] , el modelo de Gersnert [4] y Fast Fourier Transforms (FFTs) [19] .El segundo paso es la simulación de los efectos ópticos (como la refracción y reflexión) donde lideran actualmente métodos como *ray-tracing* o *reverse ray-tracing* [16] , radiosidad, mapas de reflexión y otros métodos de simulación realista en tiempo real.

2.2 Programación de la GPU

¿Qué es la GPU?

GPU es un acrónimo utilizado para abreviar Graphics Processing Unit, que significa "Unidad de Procesamiento Gráfico".

Una GPU es un procesador dedicado exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador central en aplicaciones como los videos juegos. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la CPU puede dedicarse a otro tipo de cálculos (como la inteligencia artificial o los cálculos mecánicos en el caso de los videos juegos). Una GPU implementa ciertas operaciones gráficas llamadas primitivas optimizadas para el procesamiento gráfico. Una de las primitivas más comunes para el procesamiento gráfico en 3D es el *antialiasing*, que suaviza los bordes de las figuras para darles un aspecto más realista. Adicionalmente existen primitivas para dibujar rectángulos, triángulos, círculos y arcos. Las GPU actualmente disponen de gran cantidad de primitivas, buscando mayor realismo en los efectos [15].

Shader

Debido a la existencia del hardware gráfico y su actual desarrollo, la eficiencia de las aplicaciones de visualización se pudo elevar considerablemente. Permitiendo representar escenas virtuales de mayor tamaño, complejidad y en un tiempo lo más real posible. Sin embargo, a pesar que actualmente el aumento de rendimiento ha originado un campo de investigación de muchas perspectivas con resultados relevante en cuanto a ejecutar las secuencia de visualización en un tiempo cada vez menor, se puede objetar que al mismo tiempo estos sistemas no se han desarrollado con respecto a las tecnologías de visualización y mucho menos se ha sincronizados estos dos aspectos fundamentales a la hora de representar escenas en entornos virtuales.

Hoy en día, y a través de la constante revolución del hardware gráfico se han buscado alternativas positivas para dar respuesta al problema de no poderse cambiar la estructura del hardware de aceleración gráfica, ya que cuando se crean estos hardware, los ingenieros prefijan un conjunto de instrucciones y algoritmos en el chip de video. Cada uno de estos algoritmos es acelerado por el hardware gráfico. Pero no se brinda la posibilidad de ejecutar en estos chips otras instrucciones que no sean las codificadas en el

momento de la creación del hardware. A esta estructura estática se le denomina sistema de funciones fijas (*Fixed-Function*) [11].

La alternativa actual que existe para *Fixed-Function*, limitante fundamental del hardware gráfico, para aumentar la flexibilidad gráfica del hardware de video esta en tres momentos claves del pipeline gráfico en los cuales se le han permitido introducir código que permitan ejecutar algoritmo no diseñado inicialmente en el hardware. A estos códigos se le denominan *shader*, y según su funcionamiento y lugar de ejecución se dividen en *Vertex Shader*, *Geometry Shader* y *Píxel Shader.*

Los ***Vertex Shader***, función del proceso gráfico, son los encargados de añadir los efectos especiales a los objetos de una escena 3D. Permite a los desarrolladores ajustar efectos mediante la carga de instrucciones en la memoria dedicada a él. En ellos se ejecutan las transformaciones de espacio objeto a espacio de mundo, de cámara, y finalmente se obtiene la posición en la pantalla. Además este permite controlar la posición y el contenido de dichos vértices (textura, iluminación,...).

La figura 1 muestra cómo trabaja el pipeline gráfico donde la interface de *DirectX* u *OpenGL* pasa los *vertex* al driver, la GPU recibe estos datos y/o comandos y luego el *vertex shader* transforma los vértices recibidos y realiza otras operaciones a nivel de vértices después en el geometry *shader* se realiza operaciones con entidades primitivas (líneas, triángulos o vértices). A partir de una primitiva, el **Geometry Shader** puede descartarla, o devolver una o más primitivas nuevas, el interpolador (Rasterizador) recibe la posición en la pantalla de cada uno de los vértices y genera los triángulos correspondientes. Al dibujar estos triángulos se calcula la posición de cada uno de los píxeles que conforman el triángulo.

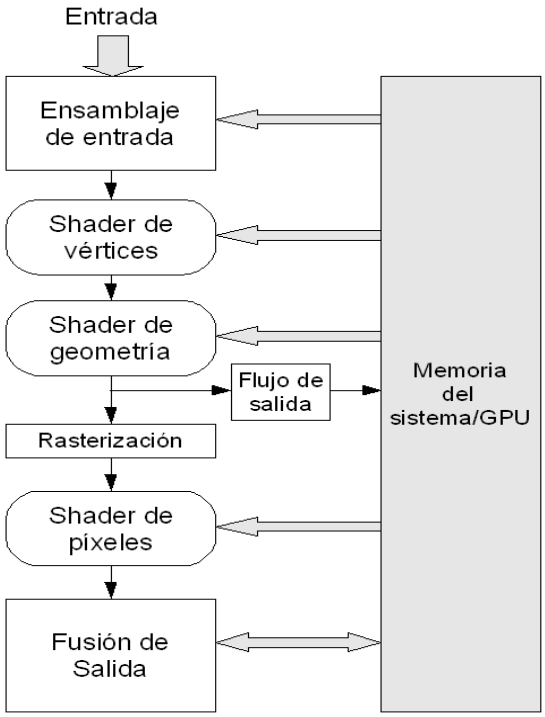

Figura 1 Pipeline Gráfico Conceptual usando Shaders

Cada uno de estos píxeles se introduce en el ***Píxel Shader*** para que este calcule el color del píxel en la pantalla. De esta forma en el *Píxel Shader* se realizan todas las operaciones a nivel de píxel como es el cálculo de la iluminación *per-píxel*, mapeo de texturas, uso de los mapas de normales para la determinación de la normal del píxel, etc.

De esta forma, el uso de los *shader* permite introducir nuevos algoritmos en el hardware de video. Dando la posibilidad de realizar avances en la visualización virtual sin la necesidad de esperar a que estos algoritmos sean codificados en el chip de video para obtener aceleración por hardware.

2.3 Iluminación de escena

Hoy en día realizar una representación gráfica de la realidad es una tarea engorrosa .Fenómenos como la iluminación influyen incisivamente en la escena donde se quieran o necesiten simular propiedades ópticas y físicas de la luz para obtener imágenes realista, de tal modo, que un observador no pueda distinguir entre la imagen real o la virtual. En la actualidad sobresaltan dos tipos de iluminación de escenas virtuales. La iluminación local en la cual no se tienen en cuenta la porción de los rayos reflejado de un objeto hacia los otros objetos que participan en la escena sino solo importa aquellos rayos que se dirigen a una superficie u ojo del observador y la iluminación global es la que considera la luz reflejada por un punto teniendo en cuenta toda la luz que llega y no solo procedente de los focos luminosos sino también de los objetos de la escena produciendo sombras, transparencia, y reflexión de un objeto en otros.

Métodos de Iluminación Global

En el intento de alcanzar un alto realismo en el proceso de simulación de escenas que ocurren en la realidad en entornos artificiales es importarte decir que es sumamente imposible representar el comportamiento de los rayos de luz que inciden un ambiente natural real por lo que ha sido un campo de investigación amplio para desarrolladores interesados en representar este fenómeno físico, mediantes métodos computacionales. Numerosos son los métodos elaborados con el propósito de simular un modelo de iluminación en escenas virtuales que representen eficientemente la realidad.

Ray-tracing (Trazado de Rayos)

El ray-tracing o trazado de rayos es un algoritmo dirigido mayormente a las interacciones especulares utilizado para generar imágenes 3D en computadoras, que llegó a ser muy popular gracias a la facilidad con el que se podía implementar y la calidad de los resultados.

La idea es que los focos emisores de luz como puede ser una bombilla o el sol, emiten rayos de luz, y estos, al colisionar con los objetos suelen hacer dos cosas diferentes:

Reflejan la luz.

Absorben la luz.

Aunque normalmente lo que ocurre es que una parte es absorbida y la otra parte es reflejada, dependiendo de las características de los objetos. De manera que los rayos inicialmente dirigidos hacia todas las direcciones empezaran a colisionar y propagarse de forma distinta a la inicial, hasta que finalmente lleguen a nuestras retinas. La suma de todos estos rayos compone la escena que se ve en ese momento.

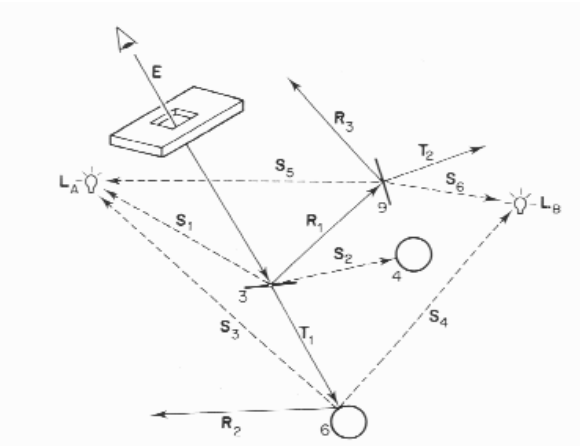

Figura 2 Trazado de rayo desde el punto de vista.

Este modelo computacional presenta limitaciones como:

- Sólo considera la reflexión especular y la refracción.
- Se considera la reflexión difusa en el rayo proveniente de la luz.
- Tendría un costo muy elevado considerar la reflexión difusa.
- La mayoría de las escenas tienen superficie con reflexión difusa.

Radiosity (Radiosidad)

La radiosity o radiosidad es un método para simular la interrelación difusa. Esto es que simula la reflexión difusa directa e indirecta procedente de las fuentes de luz y de los demás objetos de la escena. La idea es dividir los objetos de la escena en parcelas. Estas suelen ser subdivisiones triangulares o cuadradas de la malla del objeto. Cada parcela puede "expulsar" luz, la luz "expulsada" es la luz emitida mas la luz reflejada y la luz reflejada es la luz que incide sobre la parcela por su índice de reflexión.

- luz expulsada = luz emitida + luz reflejada.
- luz reflejada = luz incidente * índice reflexión.

La radiosidad progresiva se hace en varios pasos. Primero se calcula la luz que incide directamente sobre las parcelas comprobando simplemente si hay visión directa entre la fuente de luz y la parcela Ahora estas parcelas funcionan como nuevas fuentes de luz, así que se vuelve a calcular para cada parcela las fuentes de luz que le afectan directamente repitiéndose el proceso hasta que el porcentaje de la energía luminosa haya sido absorbida.

Este modelo computacional presenta limitaciones como:

- No tiene en cuenta la reflexión especular.
- Es necesario discretizar la escena en polígonos antes del cálculo.

2.4 Ópticas de las superficies líquidas

En la naturaleza, cuando la luz es emitida desde algún foco luminoso esta es refleja por innumerables objetos antes de alcanzar los ojos del espectador. Cada vez que es reflejada, una parte de la energía es absorbida por la superficie, otra parte es dispersada en direcciones aleatorias y el resto se dirige a los ojos del espectador o a otra superficie. El proceso anterior se repite hasta que la energía se reduzca a cero o el espectador perciba la luz.

Para realizar una convincente representación de superficie líquida es importante entender como se comporta la luz cuando se mira cualquier superficie líquida por ejemplo ríos, lagos y océanos atendiendo al modelo de luz presente en la escena y a su vez es importante entender los fenómenos físicos que viene aparejado a la onda luminosa. Las sustancias líquidas específicamente el agua por generalidad no se presenta de una forma transparente presentando un índice de refracción [20] diferente a otros elementos naturales como el vidrio, el aire y otros por lo que los métodos de representación de estos elementos en un entorno virtual servirá de base para el entendimiento del fenómeno de reflexión y refracción para la representación de superficies líquidas transparente (el agua).

Reflexión y Refracción

En la naturaleza si un rayo de luz que se propaga a través de un medio homogéneo incide sobre la superficie de un segundo medio homogéneo, parte de la luz es reflejada y parte entra como rayo refractado en el segundo medio, donde puede o no ser absorbido. La cantidad de luz reflejada depende de la relación entre los índices de refracción de ambos medios. El plano de incidencia se define como el plano formado por el rayo incidente y la normal (es decir, la línea perpendicular a la superficie del medio) en el punto de incidencia. El ángulo de incidencia es el ángulo entre el rayo incidente y la normal. Los ángulos de reflexión y refracción se definen de modo análogo. Las leyes de la reflexión afirman que el ángulo de incidencia es igual al ángulo de reflexión, y que el rayo incidente, el rayo reflejado y la normal en el punto de incidencia se encuentran en un mismo plano.

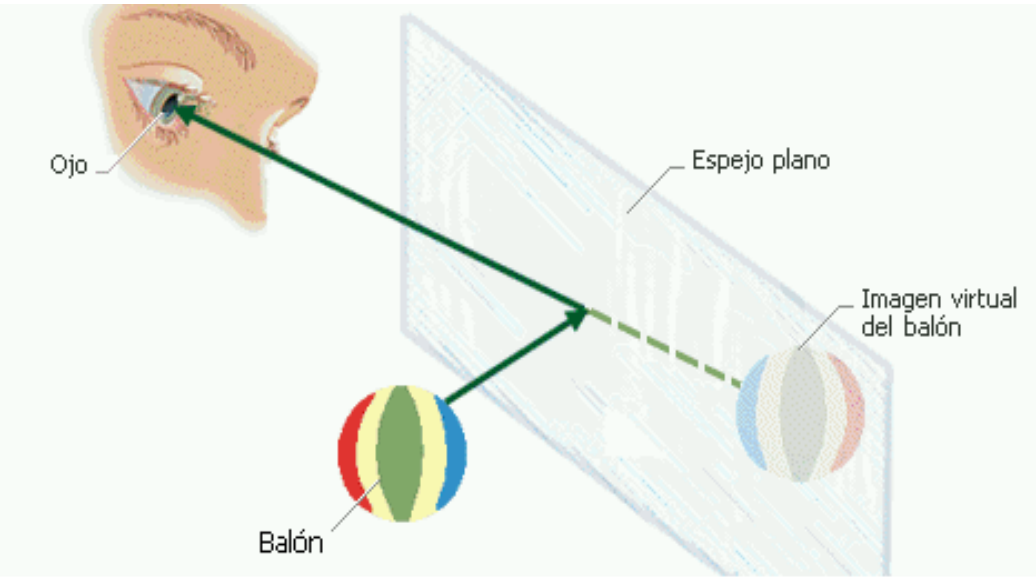

Figura 3 Reflexión en espejo pulido

Si la superficie del segundo medio es lisa, puede actuar como un espejo y producir una imagen reflejada (figura 5). En esta, la fuente de luz es el objeto A; un punto de A emite rayos en todas las direcciones. Los dos rayos que inciden sobre el espejo en B y C, por ejemplo, se reflejan como rayos BD y CE. Para un observador situado delante del espejo, esos rayos parecen venir del punto F que está detrás del espejo.

De las leyes de reflexión se deduce que CF y BF forman el mismo ángulo con la superficie del espejo que AC y AB. En este caso, en el que el espejo es plano, la imagen del objeto parece situada detrás del espejo y separada de él por la misma distancia que hay entre éste y el objeto que está delante.

Si la superficie del segundo medio es rugosa, las normales a los distintos puntos de la superficie se encuentran en direcciones aleatorias. En ese caso, los rayos que se encuentren en el mismo plano al salir de una fuente puntual de luz tendrán un plano de incidencia, y por tanto de reflexión, aleatorio. Esto hace que se dispersen y no puedan formar una imagen.

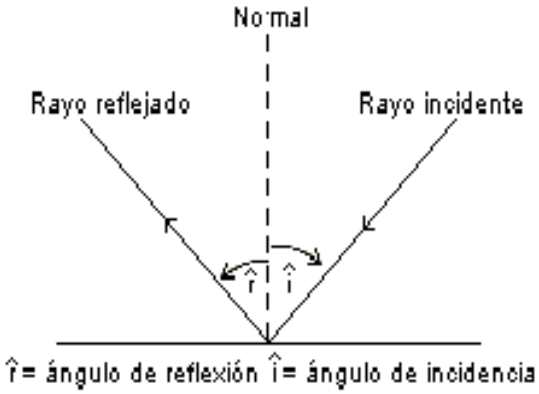

Figura 4 Leyes fundamentales de la reflexión

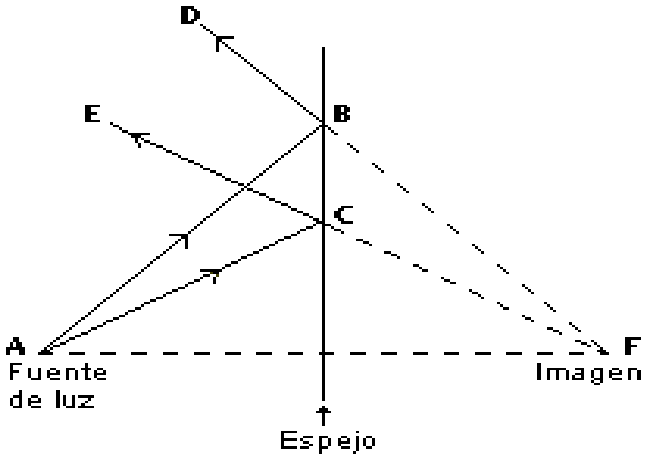

Figura 5 Reflexión en un espejo plano

El rayo incidente, la normal y el rayo reflejado se encuentran en un mismo plano. El ángulo de incidencia es igual al ángulo de reflexión.

La reflexión de la luz puede ser de dos tipos dependiendo de la naturaleza de la superficie de separación, a saber: especular (como en un espejo) o difusa (cuando no se conserva la imagen, pero se refleja la energía). Además, si la superficie de separación es entre un medio dieléctrico y uno conductor, o entre dos medios dieléctricos, la fase de la onda reflejada podría, o no, invertirse.

Reflexión Especular y Difusa

En la realidad, la reflexión difusa y la reflexión especular son generadas por el mismo proceso exacto de dispersión de la luz. La difusión es dominante en una superficie que tiene una pequeña escala de rugosidad en ella, con respecto a la longitud de onda, de forma que la luz se ve reflejada en muchas direcciones por cada pequeño fragmento de superficie, con cambios muy pequeños en el ángulo de la superficie. Por otro lado, la reflexión especular, predomina en una superficie que es suavizada, con respecto a la longitud de onda. Esto implica que la dispersión de los rayos de cada punto de la superficie será direccionada en su mayoría en la misma dirección, más que al ser dispersada de forma difusa. Es

simplemente un problema de la escala de detalle. Si la rugosidad de la superficie es mucho menor que la longitud de onda de la luz incidente parecerá plana y actuará como un espejo.

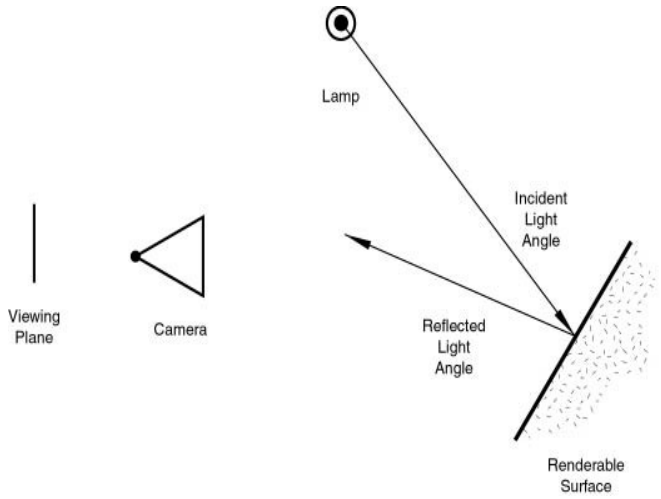

Figura 6 Reflexión Especular

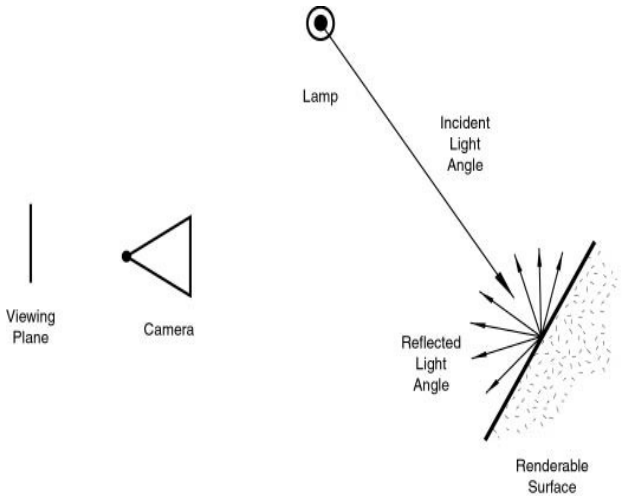

Figura 7 Reflexión Difusa

Es importante incidir especialmente en que el fenómeno de la reflexión especular discutido aquí no es la reflexión que se ve en un espejo, pero sí los destellos de luz que se ven en una superficie pulida. Para conseguir reflexiones como las de un espejo debería usar un trazado de rayos antes visto, pero esto puede producir superficies parecidas a un espejo de forma convincente, mediante una aplicación cuidadosa de texturas.

Refracción de la luz

Este fenómeno ocurre cuando un rayo de luz pasa por la interfaz entre dos medios transparentes tales como aire y agua, haciendo un ángulo θ_1 con la normal, es desviado de su dirección original de tal forma

que, dentro del segundo medio, hace un ángulo θ_2 con la normal. La relación entre estos ángulos está

dada por la **Ley de Shell**, o de Descartes, y se escribe como,

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

Ecuación 1 Ley de Shell

Donde n_1 y n_2 son los llamados índices de refracción de los medios 1 y 2, respectivamente. El índice de

refracción [20] es una propiedad óptica característica de los materiales, como lo son la densidad, resistividad, calor específico, coeficiente de expansión térmica, etc., y se define a través de la relación,

$$c = {}^{v}/_{n}$$

Ecuación 2 Rapidez de la Luz

Aquí $\mathcal{C}$ es la rapidez de la luz en el vacío y $\mathcal{V}$, la rapidez de la luz en el medio cuyo índice de refracción es $\mathcal{n}$. Mientras mayor es el índice de refracción, menor es la rapidez de propagación de la luz en el medio y mayor su desviación.

Cuando un haz de luz pasa de un medio como aire a agua, se desvía acercándose a la normal porque el índice de refracción del agua es mayor que el del aire. En el caso contrario, es decir, si pasa de agua a aire, se aleja de la normal. Por lo tanto se afirma que $n_2 > n_1$. El fenómeno de refracción de un rayo luminoso es la consecuencia del cambio en la rapidez de la luz en los diferentes medios transparentes por los cuales viaja el rayo.

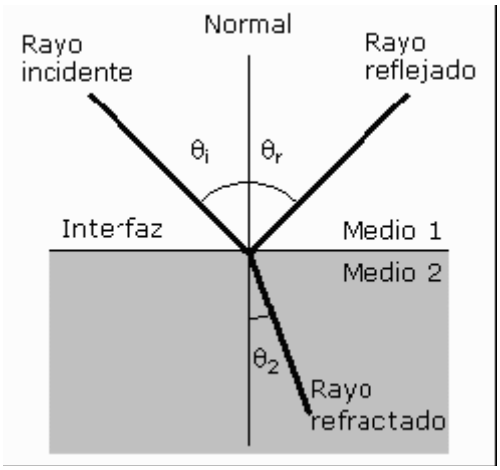

Figura 8 Reflexión y Refracción

Dispersión de la luz

La dispersión es el fenómeno de separación de las ondas de distinta frecuencia al atravesar un material. Todos los medios materiales son más o menos dispersivos, y la dispersión afecta a todas las ondas.

Cuando la luz blanca, compuesta por ondas de todas las frecuencias dentro de la gama visible, pasa a través de un bloque de vidrio, los diferentes colores son refractados o desviados en distinta medida. Si los lados del bloque no son paralelos, los diferentes colores de la luz se propagan con ángulos distintos, produciendo un espectro.

La dispersión se debe a que la velocidad de una onda depende de su frecuencia. Por ejemplo, las ondas luminosas de diferente longitud de onda tienen velocidades de propagación distintas en el vidrio, por lo que son refractadas en diferente medida.

2.5 Algoritmos de Reflexión y Refracción

En el mundo, en dependencia de la complejidad del entorno virtual o juego que se quiera realizar, en un sistema de realidad virtual de bajo nivel de realismo por lo general se utiliza para la reflexión una técnica simple para representar la geometría reflejada al revés en un plano sin perturbaciones, después de usar un plano de truncamiento (clipping plane) para mostrar solamente la reflexión sobre la superficie. Lo que significa que el observador que se encuentra encima del agua vera los objetos reflejado en el agua, si se encontrara por debajo de esta ya no vería la reflexión ya que con el plano de truncamiento se corta la geometría que está por debajo del agua y solo se mostrará lo que esté por debajo de esta.

Hay otra manera de hacer la reflexión de un entorno determinado en el agua y es utilizando la técnica del mapeado de reflexión (environment mapping) un método eficaz para simular superficies que reflejan por medio de una imagen de textura pre-computada donde la textura es usada para almacenar la imagen de el ambiente que esta alrededor del objeto que se va representar.

Por lo general existen varias maneras de almacenar el ambiente circundante, las más comunes son el mapeado de un ambiente esférico (Spherical Environment Mapping) en cual una solo textura contiene la imagen de cerco o el mapeado de ambientes cúbicos (Cubic Environment Mapping) en el cual el

ambiente se revela sobre las seis cara de un cubo por lo que va ha ser almacenado en seis texturas cuadradas.

Spherical Environment Mapping

Este método implica tener una esfera texturizada infinitamente lejana del plano en que la va a reflejar el ambiente o entorno. Creando una textura esférica mediante la vía de *prerendering* como el trazado de rayo, esta textura se le aplica a una esfera hueca y los colores del *texel* son determinados calculando los vectores del objeto a los *texels* en el mapa de ambiente. Esta técnica puede producir los resultados que son superficial similares a ésos producidos por el Ray-tracing, pero incurre en menos de un funcionamiento golpeado porque todos los colores de los puntos que se referirán se saben de antemano, así que todos lo que tiene que hacer es calcular los ángulos de la incidencia y de la reflexión.

Hay algunas limitaciones en brillo del mapeado esférico debido a la naturaleza de las texturas usadas para el mapa. El mapeado de cubo fue desarrollado para abordar esta limitación esto si se hace y se filtra correctamente por lo que son el sucesor obvio a los mapa alcaico de la esfera método que en la contemporaneidad no es muy práctico para los gráficos de video juego.

Cubic Environment Mapping

Esta es una técnica que utiliza un cubo para que el plano donde se visualizara la reflexión refleje el ambiente alrededor de este generalmente se hace con el mismo skyBox que se coge para las representaciones de los ambiente al aire libre. Aunque esto no garantiza una reflexión verdadera ya que los objetos que se encuentran próximos al plano no serán considerados en la reflexión, el efecto deseado se alcanza generalmente.

Casi siempre esto se hace determinado el vector al que el plano está viendo. El rayo de la cámara es reflejado en la normal de la superficie donde el vector de la cámara interseca al objeto. Esto da lugar al rayo reflejado que entonces se pasa al mapa del cubo dando lugar al efecto de reflexión deseado en el plano.

Esta clase de acercamiento es más eficiente que el acercamiento clásico del trazo de rayo de computar la reflexión exacta tirando un rayo y después de su trayectoria ópticamente exacta, pero debe ser observado que éstas no siempre llegan hacer reflexiones verdaderas. Otra ventaja importante es que esta es la única manera de crear reflexiones de fondos del mundo real en objetos sintéticos. Una desventaja típica de esta técnica es la ausencia de reflexiones del uno mismo: no se puede ver cualquier parte del objeto reflejado dentro de la reflexión generada por el cubemap.

Estos algoritmos de reflexión tienen la ventaja de que su implementación es muy simple y son aconsejables en aquellas aplicaciones donde la representación de los efectos de reflexión y refracción en planos de agua no se necesite de un alto realismo y se quiera mostrar la reflexión de un entorno determinado en un Lago. En cambio para lograr un alto realismo de los fenómenos de reflexión y refracción en aplicaciones de realidad virtual se propone la técnica donde se representa estos efectos ópticos de reflexión/refracción mediantes *shader*.

2.6 Algoritmos para la animación del agua

Como se menciona en epígrafes anteriores para lograr una visualización realista de la superficies de los Lago y Océanos se tiene en cuenta un aspecto fundamental para la misma que son los efectos ópticos que se observan a primera vista como son la reflexión y refracción y las perturbaciones ocurridas en la misma.

En la actualidad existen diversos algoritmos las cuales se pueden se referencia para lograr perturbaciones realista en la superficie de un cuerpo de agua. En este trabajo se harán mención de dos de los algoritmos más usado en el mundo para dar solución a este problema.

Fast Fourier Transforms (FFT)

Es el algoritmo donde se centra la animación del mar u océano. Este algoritmo es explicado y detallado en [26].Este modelo no está basado en cualquier modelo físico, pero usa el modelo estadístico basado en la observación de grande cuerpos de agua como el mar o el océano por lo que es muy usado en la vida real por ejemplo fue el modelo tomado en películas “ Titanic” y ” El mundo acuático” .

En este modelo estadístico para cuerpo de agua como el océano, la altura de la onda es una variable al azar en posición horizontal en el tiempo $h(X,t)$. Descompone el campo de altura de la onda en un

sistema de onda con diferentes amplitudes y fases. Mientras que el mismo modelo provee una herramienta para general esas amplitudes y fases. Se utiliza la inversa de FFT para evaluar rápidamente la suma.

FFT es una versión rápida de discreta transformación de Fourier (FT), es decir la transformación de Fourier que muestrea la entrada de los puntos regularmente puesto. La descripción de regular ambas transformaciones, la FT y FFT son una interesante característica y los algoritmos se pueden ver como funcionan en [26].

FFT permite evaluar la suma rápidamente de la siguiente forma:

$$h(X,t) = \sum_{K} \tilde{h}\ (X,t) e^{iKX}$$

Ecuación 3 Sumatoria de la altura de la onda

Donde X es la posición horizontal de punto de la altura que estamos evaluando. $\vec{K}$ es el vector que

apunta a la dirección de viaje de la ola dada, con la magnitud K que depende de la longitud de onda λ.

$$k = \frac{2\pi}{\lambda}$$

Ecuación 4 Altura de onda

El valor de $h(X,t)$ es un número complejo que representa la amplitud y fases de la ola $\mathcal{K}$ en un tiempo t. Ya que se usa la discreta transformación de Fourier hay solo un número finito de ecuaciones y posiciones que entra en las ecuaciones. Si se toma que S es el campo de altura que se toma para animar el cuerpo de agua y **r** la resolución de la reja .Se puede escribir que:

$$\mathcal{K} = (k_X, k_Z) = (2\pi n/S, 2\pi m/S)$$

Ecuación 5 Fase de la ola

Donde n y m son valores enteros mayores igual $-r/2$ y menores que $r/2$. Para renderizar el campo de altura se calcula el gradiente del campo para obtener la normal. Se podría usar el acercamiento tradicional para computar la diferencia finita entro los puntos de la rejilla pero se correría el riesgo de una aproximación pobre de la cuesta de onda con longitudes pequeña. Por lo que se evaluaría la suma de la siguiente forma:

$$\nabla h(X,t) = \sum_{\mathcal{K}} i\mathcal{K}\, \tilde{h}\,(X,t) e^{i\mathcal{K}X}$$

Ecuación 6 Sumatoria de la altura de onda con longitudes pequeñas

Ahora que se conoce como convertir el campo de números complejos que representa amplitudes y fases en un campo de altura se necesita una manera de crear amplitudes y poner fases .En el epígrafe se surgiere el espectro Phillips para las olas. Esta se define en la siguiente ecuación:

$$\mathcal{P}(\mathcal{K}) = a\frac{e^{-1/(\mathcal{K}\mathcal{L})^2}}{\mathcal{K}^4}\left|\hat{\mathcal{K}}\,\hat{\mathcal{W}}\right|^2$$

Ecuación 7 Espectro Phillips

Si se quiere tener el movimiento de la ola en una sola dirección solo se modifica el término $\left|\hat{\mathcal{K}}\,\hat{\mathcal{W}}\right|$ para

eliminar las olas que se mueven en oposición del viento.

También para mejorar las propiedades de convergencia de espectro, podemos eliminar las olas donde su longitud de onda sea muy pequeña multiplicando por el siguiente término:

$$e^{-\mathcal{K}^2\mathcal{W}^2}$$

Ecuación 8 Convergencia del espectro

Ahora hay dos pasos que hay que hacer para elaborar los datos para el FFT. Crear las amplitudes y las fases en un tiempo cero y después animar el campo. La primera parte se logra mediante la ecuación.

$$\tilde{h}_0(\mathcal{K}) = \frac{1}{\sqrt{2}}(\xi_r + i\xi_i)\sqrt{\mathcal{P}(\mathcal{K})}$$

Ecuación 9 Amplitud de la onda

Donde ξ_r ξ_i son variables independientes dibujada por un generador Gaussian de números al azar con un valor medio 0 y una desviación estándar de 1.

Se modifica el espectro para un mejor control de las forma de la ola.

$$\mathcal{P}_h(k) = \begin{cases} \mathcal{A}\frac{1}{k^4} e^{-(k\ell)^{-2}} |kw|^{\gamma(kw)} e^{-k^2\ell^2}, k > k_* \\ 0, k \leq k_* \end{cases}$$

Ecuación 10 Espectro Phillips modificado

Donde el parámetro k y ℓ nos permita cortar las ondas demasiadas largas y las cortas. El parámetro γ_+ y γ_- junto a w nos permita contralar las ondas. Fabricando diferentes γ_+ y γ_- se lograra que las olas se muevan en una dirección.

Se crea un campo de amplitudes de frecuencia dado un tiempo t independiente al tiempo anterior que puede tener valor de:

$$\tilde{h}\,(\mathcal{K},t) \;=\; \widetilde{h_0}(\mathcal{K})\,e^{i\,w\,t} + \widetilde{h_0}(-\mathcal{K})\,e^{-i\,w\,t}$$

Ecuación 11 Amplitud de la Frecuencia

Donde w es la frecuencia angular de la onda $\mathcal{K}$ que representa la velocidad con que viaja la onda por

la superficie. Como la altura que se obtiene en FFT inverso son numero verdaderos es decir su parte imaginaria es cero. Se demuestra que la ecuación quedaría:

$$\widetilde{h_0}(-\mathcal{K}) \;=\; \widetilde{h_0}(\mathcal{K})$$

Ecuación 12 Altura obtenida mediante FFT inverso

Debido a que estamos animando cuerpo de agua como el océano, mar con una alta profundidad y volumen de la frecuencia angular w se define como:

$$w = \sqrt{g\mathcal{K}}$$

Ecuación 13 Frecuencia angular

Donde g es la constante de gravedad y $\mathcal{K}$ es la magnitud del vector de la ola.

Y en caso de que la profundidad d sea constante la ecuación quedaría:

$$w = \sqrt{g\mathcal{K}\tan^{-1}(\mathcal{K}d)}$$

Ecuación 14 Frecuencia angular con profundidad constante

Las Olas agitadas

En el epígrafe anterior se propone un modelo matemático para que se logre una buena representación del proceso de deformación ocurrido en un cuerpo de agua en este caso el océano con un alto nivel de realismo. Pero no solo esta representación de las perturbaciones ocurridas en el océano es suficiente para simular en un mundo virtual los efectos que ocurren en el mundo real. En el mundo que nos rodea, espéciamele en el mar u océano, aparejado al movimiento no uniforme de las se encuentra la espuma producto de la agitación de la ola. Un sistema que pretenda adquirir en su desarrollo un alto nivel de realismo no puede enajenar este detalle.

Este efecto del océano se logra si hacen más puntiaguda la cresta de la ola. En vez de modificar el campo de altura directamente, se desplaza la posición de los puntos de la rejilla usando la ecuación:

$$\mathcal{X} = \mathcal{X} + \lambda\mathcal{D}(\mathcal{X}, t)$$

Ecuación 15 Posición de desplazamiento

Donde λ es una constante que controla la cantidad de desplazamiento y $\mathcal{D}$ es el desplazamiento de vector con FFT.

Ecuación de Navier-Stoke

En el campo de la Dinámica de Fluido en la Computación (CFD) la ecuación de Navier-Stoke (NSE) es para describir completamente el movimiento de líquidos viscosos incomprensible en el cual por lo general actúan tres tipos de fuerza.

- La Fuerza del cuerpo $(\mathcal{F}_g)$ que son las que actúan sobre el cuerpo de agua entero, se asume que es la fuerza de gravedad por lo que quedaría $\mathcal{F}_g = \rho g$ donde ρ es la densidad del cuerpo liquido.
- La fuerza de la presión $(\mathcal{F}_p)$ que es la que actúa hacia dentro y la normal de la superficie.
- La fuerza viscosa $(\mathcal{F}_v)$ que es producto a la fricción del agua con todo los elementos que se encuentro dentro de la misma.

Teniéndose todas las fuerzas que actúan en líquidos se utiliza la segunda **ley de Newton** para describir el movimiento.

$$\mathcal{F}_g + \mathcal{F}_p + \mathcal{F}_v = \rho \mathcal{A}$$

⇓

$$\rho\mathcal{A} = \rho\mathcal{G} + \nabla\mathcal{p} + \mu\nabla^2\mathcal{V}$$

Ecuación 16 Sumatoria de Fuerza

Ahora se asume que la densidad es constante y se escribe la ecuación:

$$\frac{\partial v}{\partial t} + (\nabla\,\mathcal{V})\mathcal{V} = \mathcal{G} - \frac{1}{\rho}\,\rho + \mu\nabla^2\mathcal{V}$$

Ecuación 17 Ecuación de Navier-Stoke

$$\nabla\,\mathcal{V} = 0$$

Ecuación 18 Campo de velocidad

La segunda ecuación afirma que el campo de velocidad debe tener cero divergencias, lo que implica que la masa debe conservarse. Las ecuaciones son usualmente definidas en un dominio computacional denominado ***D*** en el cual permanece el fluido, ya sea para un espacio 2D o 3D.

2.7 Lenguajes de Programación

Un lenguaje de programación se utiliza para controlar el comportamiento de la máquina, particularmente una computadora. Estos se determinan según su nivel de abstracción, forma de ejecución y paradigma de programación. Los paradigmas son un enfoque particular o filosofía para la construcción del software.

C++

El módulo se implementara en C++ debido a que la generalidad de lo proyecto con perfil o corte gráficos utilizan para el desarrollo de sus productos esta herramienta, además de ser el lenguaje de programación estudiado en la facultad #5. Las principales características de este lenguaje son el soporte para programación orientada a objetos y el soporte de plantillas o programación genérica (*templates*). Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (RTTI)

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje.

2.8 Librerías Gráficas: OpenGL y DirectX

Una librería gráfica es un software que genera imágenes en base a unos modelos matemáticos y unos patrones de iluminación, texturas y otros, con el objetivo de lograr la independencia del hardware (tanto dispositivos de entrada como de salida) e independencia de la aplicación (la librería es accedida a través de un interface único (al menos para cada lenguaje de programación) para cualquier aplicación.

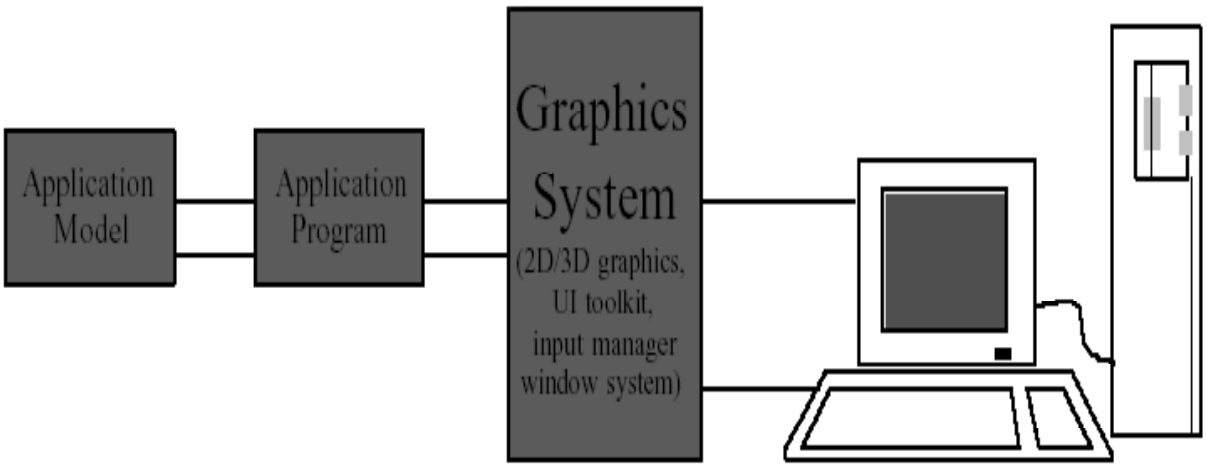

Figura 9 Diagrama de acceso a las librerías graficas

OpenGL

OpenGL es una librería gráfica que provee a los programadores de una interfaz de acceso al hardware (HW) gráfico. Es poderoso, con *rendering* a bajo nivel y una librería de software de modelamiento, disponible en la mayoría de las plataformas, con un amplio soporte de HW. Es diseñado para ser usado en cualquier aplicación gráfica, desde juegos y simuladores hasta modelaciones CAD (Computer Aided Design) [25].

DirectX

"DirectX es un intento de Microsoft de brindar un acceso directo al HW en el entorno del sistema operativo Windows, a través de un conjunto de APIs que controlan un grupo de funciones que acceden al HW o lo simulan si no existe."

Dichas funciones incluyen soporte para aceleración gráfica 2D y 3D, control de dispositivos de entrada, funciones para mezclar y probar sonidos y música, control para juegos en red y multiplayer, y control sobre varios formatos de streaming de multimedia (streaming es un método de transferencia de datos continuamente, que permite mostrar los datos antes de que el fichero entero haya sido transmitido).

Ambas APIs usan la tradicional graphics pipeline (tubería gráfica). Es el mismo pipeline que se diseñó desde las primeras computadoras gráficas, que se ha ido modificando de acuerdo a los avances de HW aunque sin cambiar la idea básica.

Ambas APIs describen los vértices como un grupo de datos consistentes en coordenadas en el espacio que definen la localización del vértice. Las primitivas gráficas (puntos, líneas, y triángulos) están definidos como un grupo ordenado de vértices. Sin embargo, la diferencia entre las APIs está en cómo los vértices son combinados para formar las primitivas: cada uno lo maneja diferente.

2.9 Lenguajes de Programación de Shader

OpenGL Shading Language

OpenGL Shading Language (GLSL) es un lenguaje de alto nivel diseñado específicamente por el entorno OpenGL. Este lenguaje permite aplicaciones para hardware gráfico. Contiene funciones que permiten expresiones abreviadas de algoritmos gráficos de manera que sea natural para programadores con experiencia en C y C++.

GLSL incluye tipos escalares, vectores y matrices; estructuras y arreglos; tipos de muestras para acceder a texturas; un tipo de dato que define entradas y salidas; constructores para inicialización y conversión; y operadores y controles declarados justo como en C y C++.

Ofrece opciones avanzadas en el diseño de gráficos al proporcionar acceso de alto nivel a las características programables de los procesadores de gráficos modernos, lo que representa un gran paso adelante en la creación de gráficos 3D fotos realistas en tiempo real.

Posee implementaciones en UNIX, Windows, Linux y otros sistemas operativos. Esta amplia compatibilidad permite a los desarrolladores mover fácilmente sus trabajos entre los principales sistemas operativos comerciales y las plataformas hardware.

Entre sus principales características se encuentran:

- Posibilidad de crear sombreados asociados al aspecto (fragmentos) de la geometría (vértices) de un objeto 3D.
- De una sola vez pueden aplicarse sombreados sobre diferentes renderizado y generar distintos resultados que se almacenan en buffer.
- La aplicación de texturas no está condicionada por su tamaño que, a diferencia de lo que ocurría en el pasado, no tiene por qué ser potencia de dos. De esta forma ahora se soportan texturas rectangulares y se reduce el consumo de memoria.

Se pueden aplicar patrones (*stencil*) sobre las dos caras de las primitivas geométricas, mejorando el rendimiento en el volumen sombreado y en los algoritmos de *renderizado* de geometría sólida.

High Level Shading Language

High Level Shading Language (HLSL) está desarrollado en base a las funciones de C y C++. Este lenguaje tiene muchas características de estos lenguajes estándares, tales como funciones, expresiones, declaraciones, tipos de datos estándares, tipos de datos creados por el usuario e instrucciones para el procesador.

HLSL soporta instrucciones para escribir expresiones matemáticas. Como otros lenguajes gráficos, las expresiones matemáticas son más eficientes con vectores y matrices. Este lenguaje sigue muchas reglas e instrucciones de C, así como otras propias para hacer la programación gráfica más intuitiva y compacta.

C para gráficos

El C para gráfico o como es más conocido Cg es un lenguaje de programación basado en estándares y diseñado para aprovechar los enormes avances experimentados por los procesadores gráficos programables que soportan DirectX y OpenGL. Como resultado, los desarrolladores pueden reutilizar sus efectos especiales en múltiples plataformas, que incluyen consolas de juegos, PC y Macintosh.

Cg ha sido desarrollado por NVIDIA en estrecha colaboración con Microsoft para garantizar total compatibilidad con DirectX 9.0 y HLSL. Además, Cg seguirá siendo compatible con futuras versiones del HLSL de Microsoft a medida que se vayan publicando.

Un lenguaje de gráficos de alto nivel resulta muy atractivo ya que permite desarrollar efectos gráficos así como impactantes aplicaciones de larga duración. Así mismo, el uso de Cg aumenta la productividad de los programadores y reduce el tiempo de desarrollo de juegos de mayor complejidad gráfica.

Basado en el nuevo lenguaje Cg estándar del sector, el compilador Cg de NVIDIA revolucionará los efectos gráficos generados por ordenador y proporcionará al escritorio un realismo cinematográfico. El nuevo lenguaje de programación de alto nivel beneficia a todo el sector:

- Los usuarios obtienen mayor realismo, impresionantes efectos visuales e interfaces gráficas de usuario más flexibles.
- Los programadores pueden reducir de forma sustancial el tiempo de desarrollo de efectos complejos.
- Los artistas y creadores de contenido digital disfrutan de fácil acceso a los recursos de la GPU.
- Los distribuidores de aplicaciones agradecerán la rápida puesta en venta y la exclusividad de los productos.

- El hardware gráfico mejorará directamente el rendimiento del software. Los desarrolladores utilizarán el kit de herramientas Cg de NVIDIA para acceder no sólo a la tecnología del compilador, sino también al resto de herramientas que optimiza el desarrollo de sombreado Cg.

2.10 El Desarrollo de Shader

En estos momentos, las herramientas para el desarrollo de GLSL y HLSL están envía a un desarrollo paulatino. Aunque existen algunas herramientas para el desarrollo del *shaders*, como Render Monkey de ATI, estas empiezan a emerger de su niñez gracias al avance creciente de las compañías encargadas a la producción de hardware y software para el desarrollo y explotación de los *shaders* [21].

Herramientas para el Desarrollo de Shader

En la actualidad existen diversas herramientas para el desarrollo de *shader* en el mundo de la programación de entornos virtuales. Los *shaders* hechos en GLSL necesitan ser creados y ser probados antes de la inyección en el uso que los utilizará. Por lo que existen varias herramientas de desarrollo de GLSL:

RenderMonkey - Creado por ATI, proporciona un interfaz para crear, para compilar y para eliminar errores de *shaders* de GLSL así como los *shaders* de DirectX. Funciona solamente en Microsoft Windows.

GLSLEditorSample – La aplicación funciona solamente debajo de Mac OS X. Permite la creación y la compilación del *shader*, pero no se ejecuta ninguna depuración. Es parte del paquete de Xcode, versiones 2.3 hacia arriba.

Quartz Composer - Un ambiente de programación visual que funciona debajo de Mac OS X. Permite la creación, la compilación y la integración del *shader* con otros parches de Quartz en su modelo de la programación visual. No se ejecuta ninguna depuración. Es parte del paquete libremente distribuido de Xcode.

Lumina - Una nueva herramienta de desarrollo de GLSL. Es independiente de la plataforma y el interfaz utiliza el cuarto de galón.

Blender - Este GLP modelado de 3D y paquete de la animación contiene la ayuda de GLSL en su motor del juego, en fecha la versión 2.41.

Shader Designer - Este GLSL extenso, fácil de utilizar IDE ha cesado la producción por TyphoonLabs. Sin embargo, puede todavía ser transferido y ser utilizado libremente. El diseñador de *Shader* viene con *shaders* del ejemplo y la documentación de la clase particular del principiante.

Demoniak3D - Una herramienta que permita cifrar rápidamente y prueban sus *shaders* de GLSL. Demoniak3D utiliza una mezcla de scripting de XML, de LUA y de GLSL para construir las escenas en tiempo real un 3d.

Capítulo 3 Soluciones Técnicas

3.1 Algoritmo para la animación del agua

Realmente se pudiera usar NSE para representar toda la dinámica del agua, pero incluso las soluciones de orden O(N)4 siguen siendo demasiado costosas a nivel de cómputo para los propósito de tiempo real y mejorar la realidad de la representación. Por lo que se propone utilizar NSE para los detalles en la superficie solamente, restringiendo el problema a dos dimensiones (2D). Según lo mencionado antes la base de la animación será el algoritmo del FFT-agua. Esto provee de olas grandes, usadas por la geometría real y olas pequeñas para la interacción de los objetos con la superficie dando una mirada realista a la simulación [19].

3.2 Algoritmos para la Reflexión y Refracción

En el capítulo anterior se hizo referencia a los algoritmos más utilizados actualmente para realizar las reflexiones ambientales a la hora de visualizar lo más real posible un Lago. En este epígrafe se dará a conocer, por las características y funcionalidades que posee, el algoritmo más adecuado según las necesidades planteadas en la introducción de este trabajo. La reflexión con un cubemaps

Desde la introducción del soporte de hardware para el cubemaps. El mapeado de entornos cúbico se ha vuelto en un método muy popular para rendir geometrías reflexivas debido a la simplicidad del algoritmo, a que ofrece resultados realistas y no exige un alto consumo de memoria por lo que es la técnica que más se ajustara al problema planteado en el capitulo anterior.

Como se puede apreciar en un entorno natural real cuando uno observa de cerca los grandes cuerpos de agua como océanos, mar y lagos no se no presenta de forma transparente, de tal forma que no se pueda ver el fondo y no se distingue bien la reflexión. Esto es debido a factores externos que influyen considerablemente en las coloraciones que estos pueden tomar como por ejemplo la profundidad,

volumen, la luz o las partículas que se encuentran dispersas dentro de estas. Este fenómeno se conoce como efecto Fresnel.

La ecuación de Fresnel describe la reflexión y refracción que ocurre en el límite del material como una función que depende del ángulo de incidencia, longitud de onda de la luz y el índice de refracción del material involucrado [11].

En este trabajo se hará uso de la aproximación para radio entre la luz reflejada y la luz refractada la ecuación creada por Christophe Schlick que plantea:

$$\mathcal{F} = \mathcal{f} + (1 - \mathcal{f})(1 - \mathcal{VN})^5$$

Ecuación 19 Aproximación al coeficiente Fresnel

Donde:

$$\mathcal{f} = \frac{\left(1{,}0 - \frac{n_1}{n_2}\right)^2}{\left(1{,}0 + \frac{n_1}{n_2}\right)^2}$$

Ecuación 20 Refracción del material

3.3 Lenguaje de Programación

GLSL es un lenguaje de alto nivel diseñado específicamente por el entorno OpenGL. Este lenguaje permite aplicaciones para hardware gráfico. Contiene funciones que permiten expresiones abreviadas de algoritmos gráficos de manera que sea natural para programadores con experiencia en C y C++.

Ofrece opciones avanzadas en el diseño de gráficos al proporcionar acceso de alto nivel a las características programables de los procesadores de gráficos modernos, lo que representa un gran paso adelante en la creación de gráficos 3D fotos realistas en tiempo real.

Aunque GLSL y HLSL son muy parecidos en cuanto a estructura y funcionamiento, como el módulo solo será implementado utilizando la biblioteca gráfica OpenGL, GLSL es el lenguaje con el que se implementará el *shader* a utilizar.

3.4 Herramientas de desarrollo

Las herramientas que se proponen en este epígrafe serán puestas en marcha en todas las fases de desarrollo del módulo.

Visual Studio 2003

Microsoft Visual Studio 2003 es un software desarrollado por la compañía Microsoft, entre las características fundamentales que presenta se encuentran algunas como, su capacidad de crear software profesional con rapidez, reducir los costos de funcionamiento de tecnologías de la información y además se integra con una amplia gama de aplicaciones, sistemas y dispositivos, la plataforma sobre la que se desarrollará este trabajo es el entorno de programación Visual C++, debido a que la programación correspondiente a la parte gráfica y el modelo matemático se realiza en C++,además de ser un lenguaje con posibilidad de desarrollo en plataforma libre.

Rational Rose

Es una herramienta de desarrollo del software para el Modelado Visual. Rational tiene gran ventaja para el equipo de desarrollo ya que los unifica a través del modelamiento el cual está basado en el Unified Modeling Language™ (UML). El UML es la notación estándar para arquitectura de software. Esto significa que con Rational Rose, todo el equipo puede comunicarse con un lenguaje y una herramienta. Rational Rose domina el mercado de herramientas para el análisis, modelamiento, diseño y construcción orientado a objetos. De acuerdo a International Data Corporation (IDC). Por cuatro años consecutivos IDC ha nombrado a Rational Rose como "La Herramienta Líder en Análisis, Diseño y Construcción Orientada a Objetos", con base en los ingresos del producto. Rational Rose permite visualizar, entender, y refinar los requerimientos y arquitectura antes de enfrentar el código. Esto permite, evitar esfuerzos desperdiciados en el ciclo de desarrollo. El modelo arquitectónico puede ser rastreado hacia el modelo de procesos de negocios y los requerimientos de sistema.

Esta herramienta permite especificar, analizar, diseñar el sistema antes de codificarlo. Tiene varias características que lo distinguen como son el de chequear la sintaxis UML, mantiene la consistencia de los modelos del sistema del software, genera la documentación automáticamente, la generación de código a partir de los modelos, permite la ingeniería inversa (crear modelo a partir código) entre otras.

3.5 Herramientas para el desarrollo de Shader

Anteriormente se hablo brevemente sobre las herramientas para el desarrollo de *shader* en GLSL. Debido a sus características y ventajas, la herramienta que se selecciono fue RenderMonkey que es un entorno de desarrollo rico del *shader* para los programadores, ya que con su uso se facilita la creación de colaboración de los efectos en tiempo real del *shader*.

Con esta herramienta se provee de un redactor completamente equipado, se proporciona la regeneración visual inmediata para el efecto que se está desarrollando asegurándose que los errores de la programación sean identificados temprano.

RenderMonkey incluye el apoyo total para idiomas del último shading de DirectX® (9.1), de OpenGL® (2.0), y de OpenGL ES® (2.0). RenderMonkey también incluye una gran cantidad de efectos de muestra, que están libremente disponibles para ser utilizados y ser modificado para su uso en requisitos particulares del desarrollador.

Capítulo 4 Construcción de la Solución Propuesta

4.1 Objeto de estudio

El objeto de estudio de este trabajo es dar solución a uno de los principales problemas que presenta la facultad # 5, la inexistencia de un módulo capaz de representar visualizaciones de Lagos y Océanos; que sean, además, eficiente en cuanto a la velocidad de *render*, en el realismo de la escena, que utilice adecuadamente las potencialidades de las tarjetas gráficas y que estén dirigidas a la biblioteca gráfica OpenGL.

Anteriormente se hizo referencia, guiado por el estudio realizado, al algoritmo que se considera como el adecuado para realizar la deformación de un plano de agua para alcanzar los efectos visuales de un Océano y lograr el realismo deseado, no se la dará implementación pero la estructura queda documentada de tal manera que podrá ser diseñada y representada posteriormente de la misma manera que serán simulado el efecto visual de reflexión en un Lago.

Obtener un módulo capaz de visualizar eficientemente el efecto de reflexión que ocurre en un Lago y Océano es el campo de acción de este trabajo. Actualmente la facultad no cuenta con un módulo que visualice dicho efecto en un Lago u Océano con técnicas eficientes para el render y que muestren un alto realismo en la simulación del mismo.

Los objetivos propuestos para este trabajo y que derivan en la obtención de un módulo de visualización eficiente de Lagos y océanos son los siguientes:

- El empleo de algoritmos para la representación eficiente de los efectos ópticos ocurrido en un plano de agua (Lago u Océanos) que proveen a la escena virtual de gran realismo.
- La incorporación de técnicas eficientes para eliminar lentitud en las visualizaciones.
- El soporte de una de las dos bibliotecas gráficas mencionadas en el capítulo 1, respondiendo así a las tendencias que existen en el mundo.

4.2 Reglas del Negocio

El fichero *shader* a cargar para la representación del efecto de reflexión del agua deben ser de extensión VERT y FRAG para ambos casos, de no existir el fichero en el camino indicado, ser de otro formato, estar corrupto o estar vacío, la aplicación no se detiene, pero no se ejecuta ningún código *shader* por lo que no se obtiene el resultado esperado.

4.3 Modelo del Dominio

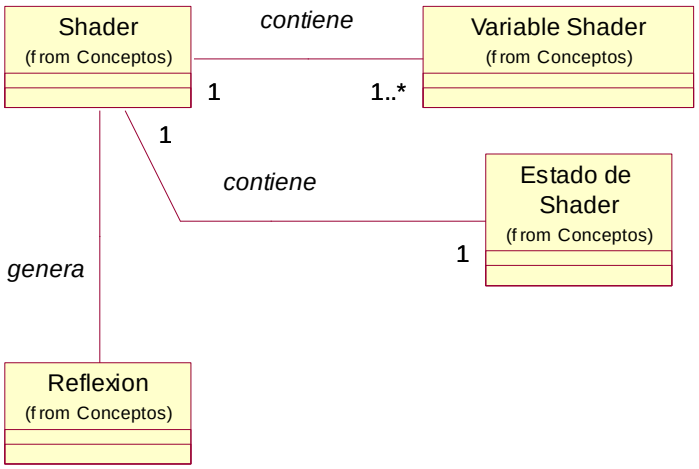

Figura 10 Modelo Conceptual

Glosario de términos del modelo del dominio

Estado de shader: Encargado de almacenar lo que devuelve **Shader**, maneja las variables controladoras. Define el estado en que se encuentra el *shader* cargado.

Shader: Encargado de manejar los ficheros con el código a interpretar y almacenar para su posterior uso.

Variables shader: Estructura encargada de interpretar las variables utilizadas en el fichero con el código ***glsl*** y enlazarlas con el programa.

Reflexión: Estructura encargada de generar el efecto óptico de reflexión a partir de una serie de variables que se le introducen. Calcula como debe quedar el reflejado.

4.4 Captura de Requisitos

A continuación se expondrán los requisitos funcionales y no funcionales del sistema.

Requisitos Funcionales

1. Crear un Cubemap
2. Cargar Texturas
3. Generar mapa de reflexión
4. Crear plano de agua
5. Crear *shader*
6. Crear objeto del *shader*
7. Crear objeto del programa
8. Cargar *shader*
9. Obtener coordenada del plano de agua
10. Obtener las coordenadas de la textura activa
11. Calcular y normalizar vector incidencia en el plano
12. Calcular constante de opacidad
13. Calcular vector de reflexión y refracción
14. Asociar el *SamplerCube* del *shader* con las coordenadas de reflexión y refracción
15. Mesclar el color de cada pixel de la escena
16. Atachar datos del fichero al *shader*
17. Compilar el *Shader*
18. Generar efecto

Requisitos no Funcionales

Usabilidad: Los futuros usuarios del sistema serán programadores con conocimientos básicos de programación gráfica y de la terminología afín. El producto debe estar concebido para que el usuario piense en qué desea hacer y no cómo hacerlo, por lo que éste requerimiento debe estar presente en alto grado en el producto final.

Rendimiento: Como aplicación de tiempo real, debe tener alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad.

Soporte: En una versión inicial deberá ser compatible con la plataforma Windows, pero debe estar preparado para que con rápidas modificaciones pueda migrar para Linux.

Hardware: Compatibilidad con cualquier tarjetas gráficas que soporte versiones mayores de 1.1 vertex *shader* y pixel *shader*.

Diseño e implementación: Debe utilizar transparentemente la biblioteca gráfica OpenGL, en su primera versión, y ser adaptable a trabajar con otras bibliotecas. Se harán llamadas a dicha biblioteca desde C++ y se usara el GLSL (OpenGL Shading Language) para el manejo de los *shader*. Se regirá por la filosofía de Programación Orientada a Objetos.

4.5 Modelo de Casos de Usos del Sistema

En esta sección se reconocen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de los requisitos funcionales anteriormente hallados, los posibles resultados de valor que le pueda brindar a sus actores, o lo que es lo mismo, los casos de uso del sistema.

Además, se seleccionan los casos de uso correspondientes al primer ciclo de desarrollo para hacerles sus especificaciones textuales en formato expandido.

Actor del sistema

Actores	Justificación
Aplicación Final	Es el que se beneficiará con las funcionalidades que brinda el módulo de clases, a groso modo: cargar desde ficheros, establecer *shader* y ejecutar el ciclo de la escena.

Tabla 1 Actores del sistema

Casos de uso del sistema

1. Crear Lago
2. Gestionar *Shader*
3. Cargar *Shader*

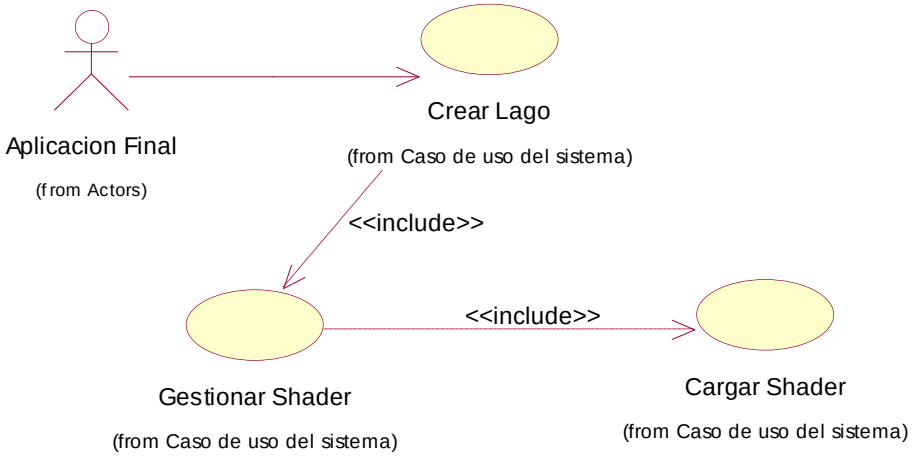

Figura 11 Diagrama de Caso de Uso

La Aplicación Final inicia el Caso de Uso (CU) Crear Lago. Este se encarga de crear el entorno donde se ubicara el Lago, llama al CU Gestionar *Shader* en el cual se crea los objetos *shader* donde se almacenarán el código fuente de los *shaders* ,llama al CU Cargar *Shader* y se crea el objeto programa donde se le atacha los objeto creados. El CU Cargar *Shader* se encarga de cargar los algoritmos con la técnica a utilizar para una real visualización del efecto de reflexión y refracción del ambiente en la escena del Lago.

Especificación de los casos de uso en formato expandido

Nombre del caso de uso	**Cargar Shader**
Actores	Aplicación Final
Propósito	Interpretar el *buffer* de datos del fichero y almacenarlos en las estructuras de datos de los *shader*.
Resumen: Se inicia cuando la Aplicación Final pasa una dirección de fichero. De ser válida la dirección se llenan los *buffers*. Se crean las estructuras necesarias para almacenar el *shader*, y se le pasa al objeto de *shader* definido anteriormente finalizando el caso de uso.	
Referencias	R9, R10, R11, R12, R13, R14, R15 CU3.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita cargar el *shader* pasando como parámetro la dirección del fichero.	
	2- Se cargan las variables a utilizar en el *vertex program*.
	3- Se obtienen las coordenadas del espacio mundo del plano del agua.
	4- Se obtienen las coordenadas de la textura activa.
	5- Se calcula el vector incidencia y se normaliza.
	6- Se normaliza la normal al plano de agua.
	7- Se calcula la constante de opacidad del

	agua.
	8- Se calcula el vector de refracción y reflexión
	9- Se cargan las variables a utilizar en el *fragment program*.
	10- Se asocia el *SamplerCube* del *shader* con las coordenadas de reflexión y refracción calculadas anteriormente.
	11- Se mescla el color de cada pixel de la escena.
Pos condiciones	Nuevo *shader* cargado.

Tabla 2 Especificación del caso de uso Cargar Shader

Nombre del caso de uso	**Gestionar Shader**
Actores	Aplicación Final
Propósito	Crear nueva estructura encargada de almacenar e interpretar los *shader*.
Resumen: Se inicia cuando el Aplicación Final solicita la creación del efecto de reflexión y refracción del ambiente de un Lago. Se crea un objeto del *shader*. Se inicializan los parámetros para la posterior adición de los *shader*. Se llama al caso de uso "Cargar *Shader*" y se le asigna al objeto de *shader*	
Referencias	R6, R7, R8, R16, R17, CU2.

Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita la adición del efecto de reflexión del Lago.	
	2- Se crea el objeto del *Shader*.
	3- Se crea el objeto del Programa.
	4- Se inicializan los parámetros.
	5- Se llama al CU “Cargar *Shader*”
	7- Se adicionan las variables controladoras al objeto del programa.
	8- Se le asigna el objeto al objeto del programa.
Pos condiciones	Nuevo objeto *shader* asignado al objeto del programa.

Tabla 3 Especificación del caso de uso Gestionar Shader

Nombre del caso de uso	**Crear Lago**
Actores	Aplicación Final
Propósito	Crear un lago donde se visualice los efectos de reflexión y refracción.
Resumen: Se inicia cuando la Aplicación Final solicita la creación de un Lago. Se realiza las operaciones de dibujo del ambiente asociado al Lago. Se llama al caso de uso "Gestionar *Shader*" y se renderea todo junto finalizando el caso de uso.	
Referencias	R1, R2, R3, R4, R5, R18, CU1.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita la creación de un Lago	
	2-Cargar texturas
	3- Se crea el Cubemap.
	4- Generar mapa de Reflexión
	5-Se crea el plano de Agua.
	6- Se llama al CU "Gestionar *Shader*".
	7- Se crea la superficie reflejada
Pos condiciones	Se crea una superficie reflejada.

Tabla 4 Especificación del caso de uso Crear Lago

Capítulo 5 Diseño del Sistema

Introducción

La primera parte del capítulo encierra los diagramas de clases de análisis del sistema propuesto. Su objetivo es brindar una primera visión de las posibles clases del diseño a un alto nivel, pero donde se dejan ver sus responsabilidades y relaciones, aún sin el mayor rigor entre ellas.

A continuación se presentan los diagramas de clases como resultado del refinamiento de las etapas anteriores. Se presentan además los diagramas de secuencia de la realización de los CU.

5.1 Diagrama de clases del Análisis

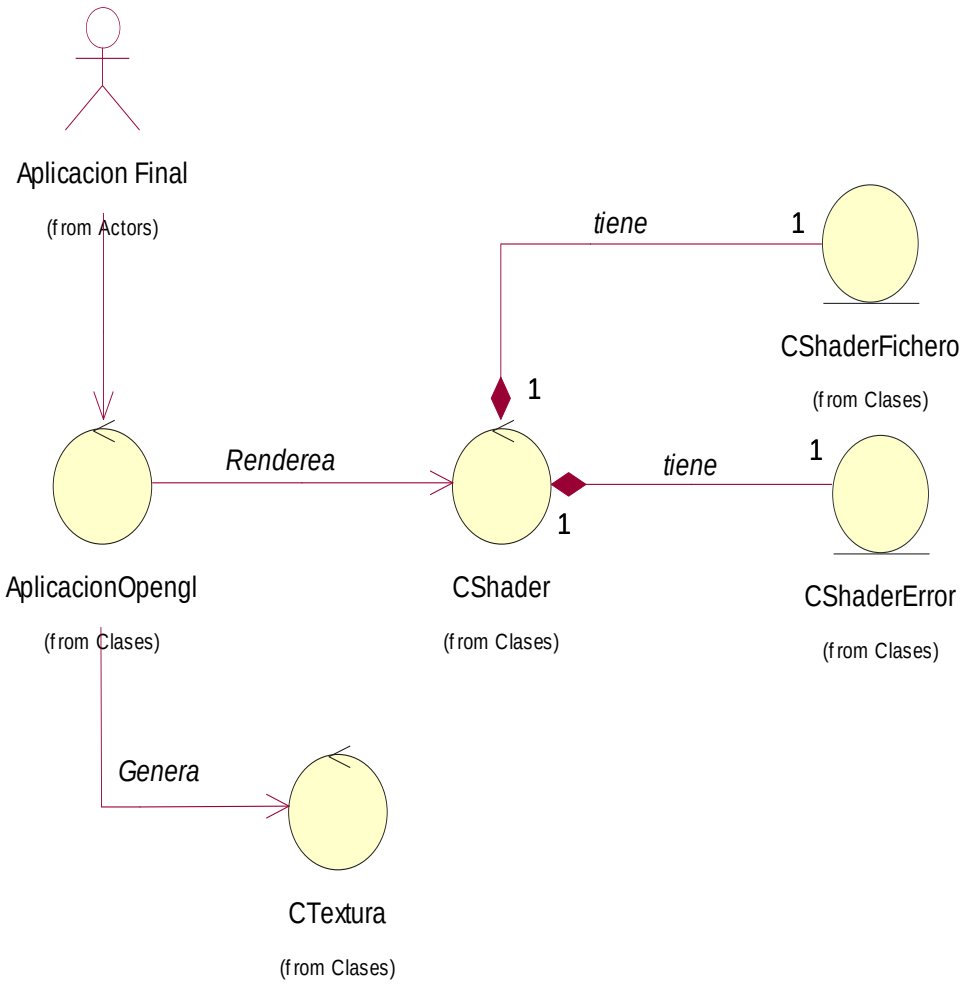

Figura 12 Diagrama de clases del Análisis

5.2 Diagrama de clases de Diseño

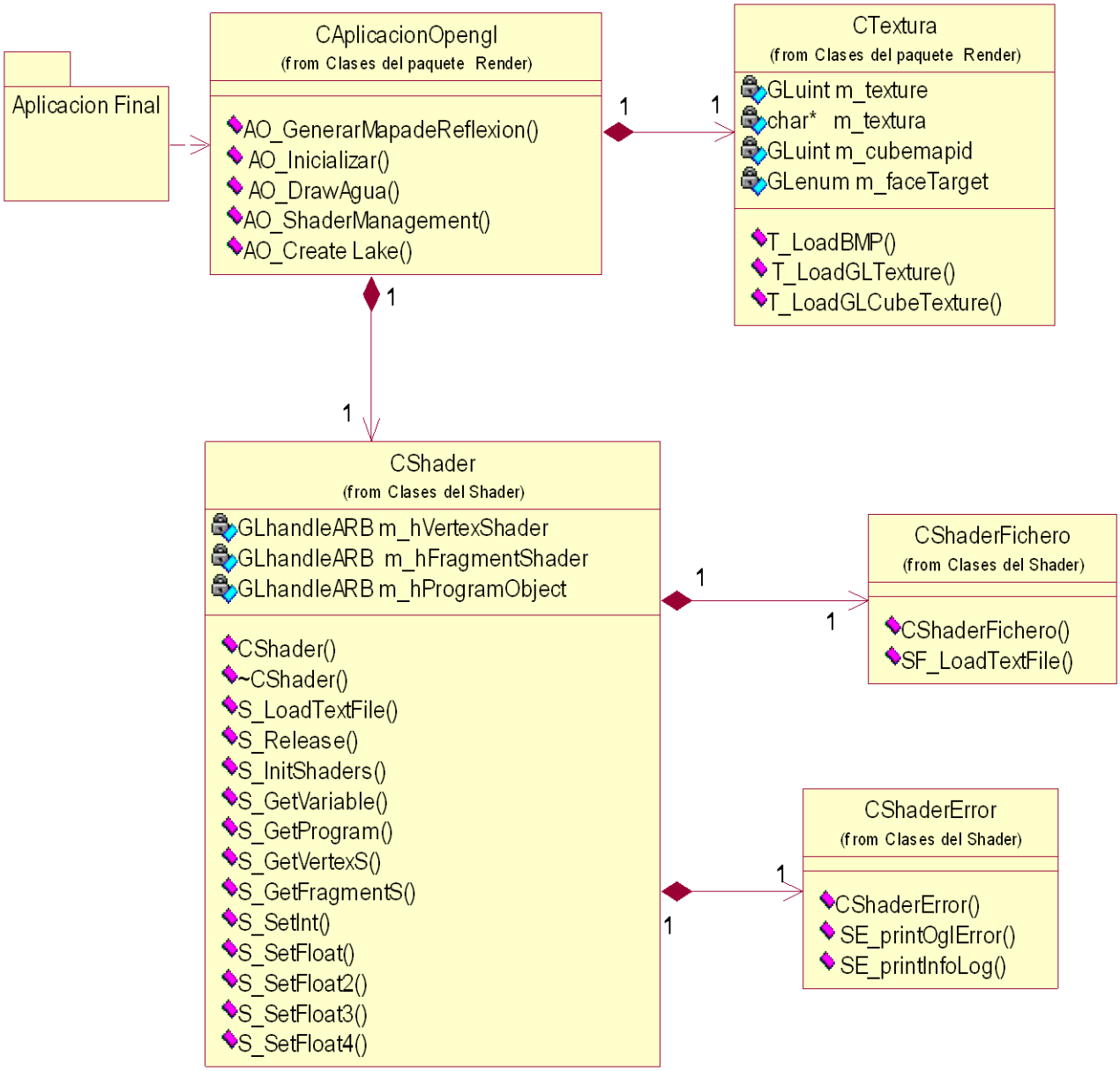

Figura 13 Diagrama de Clase de Diseño

Para lograr un mejor entendimiento del Diagrama de clase del diseño y lograr una alta modularidad se agruparon las clases en paquete; el paquete Render encapsula la clase CAplicacionOpengl y CTextura del demo y el paquete *Shader* encapsula las clases cuya funcionalidad son las de manipular o manejar los *shader*. El paquete Aplicación Final encapsula las clases de aquellas aplicaciones futuras que se beneficiaran con la visualización de la superficie del Lago.

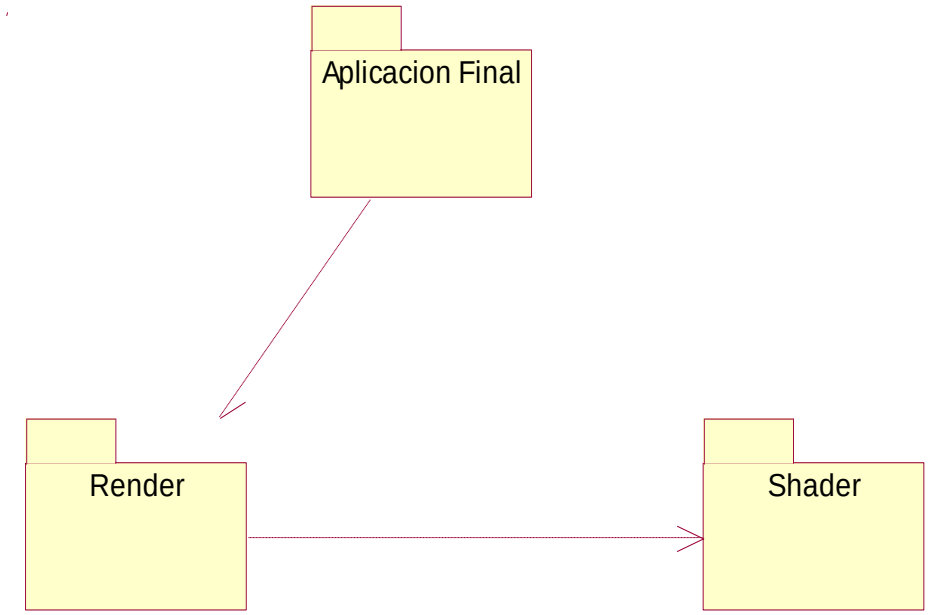

Figura 14 Diagrama de Paquete del Diseño

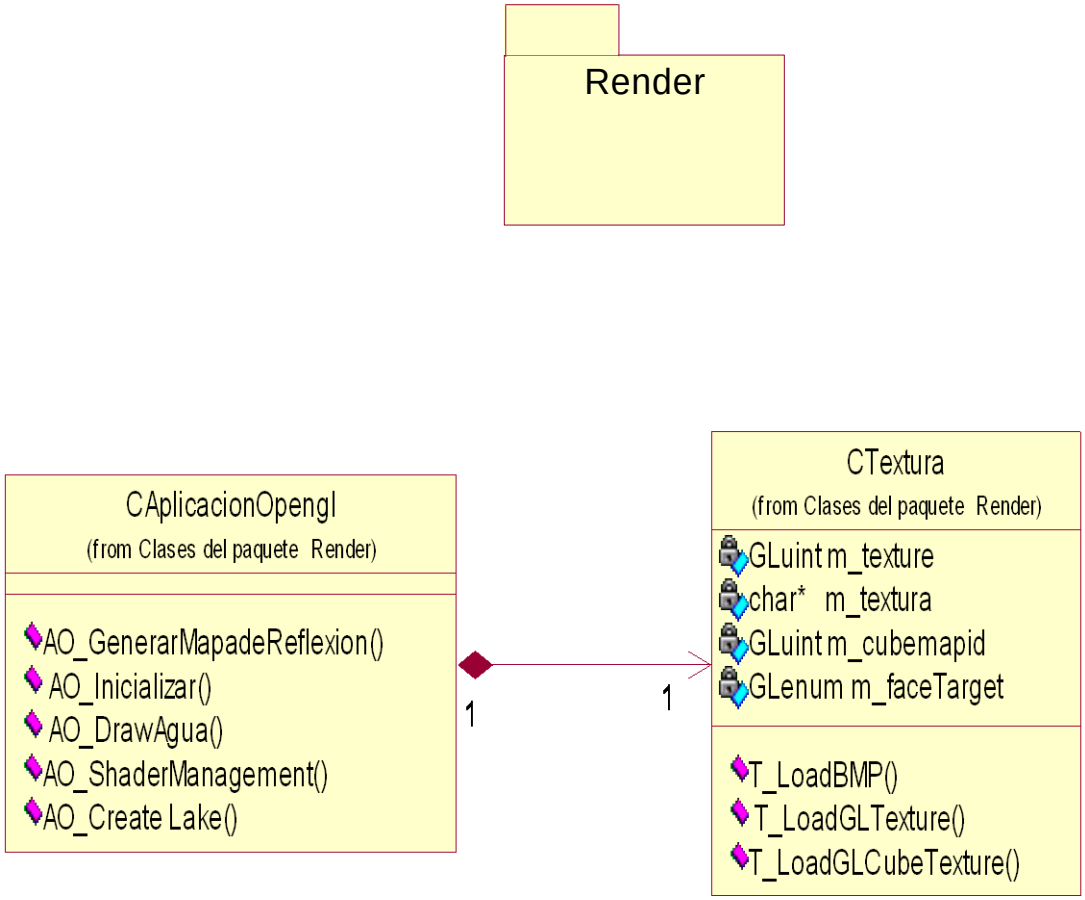

Figura 15 Diagrama de Clase del Paquete Render

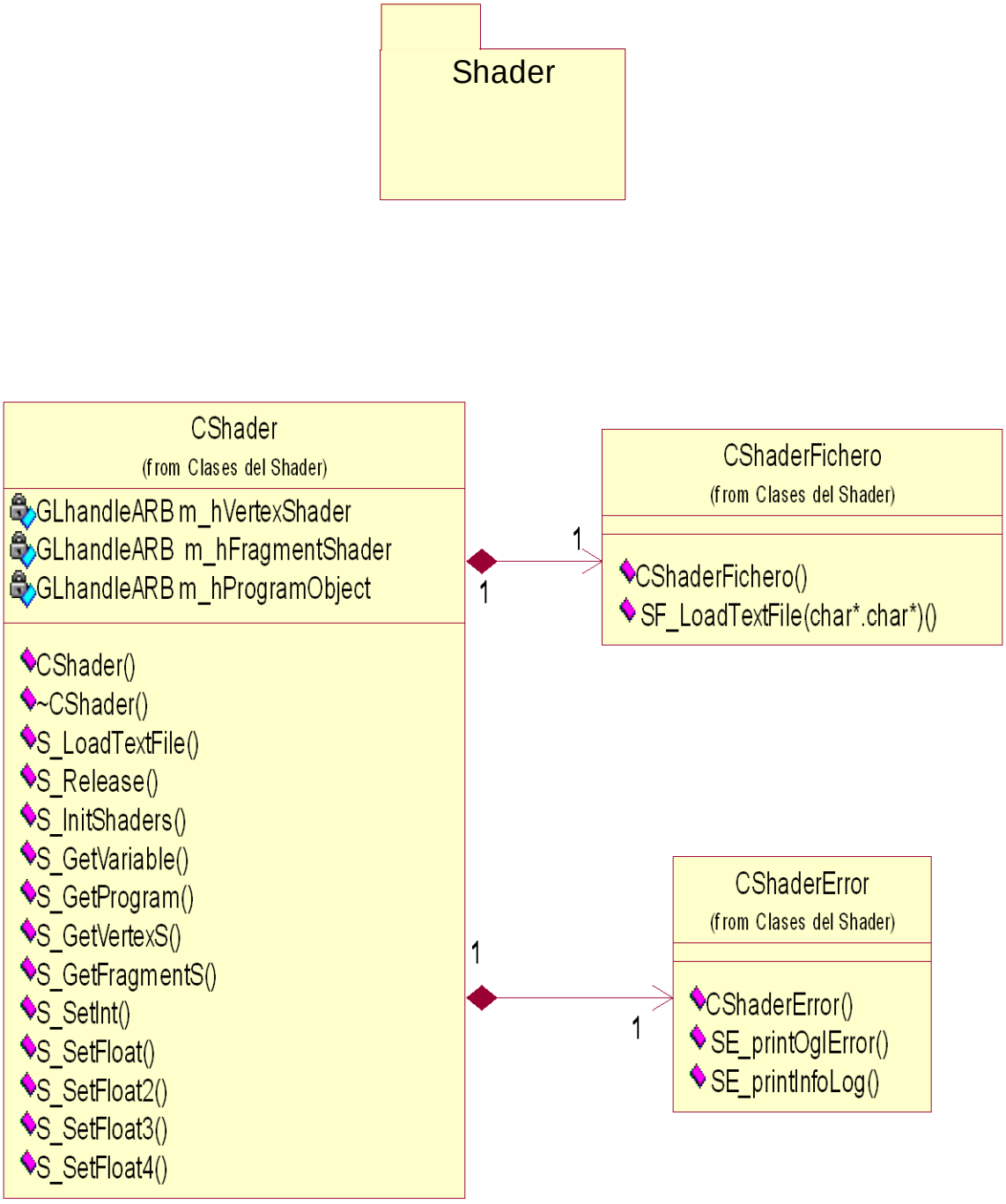

Figura 16 Diagrama de Clase del Paquete Shader

5.3 Descripción de las clases del Diseño

Nombre: CShader	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_hVertexShader	GLhandleARB
m_hFragmentShader	GLhandleARB
m_hProgramObject	GLhandleARB
m_fichero	CShaderFichero*
m_error	CshaderError*
Para cada responsabilidad:	
Nombre:	S_Release
Descripción:	Función encargada de descargar o liberar toda la memoria para los datos *shader*.
Nombre:	S_InitShaders
Descripción:	Función encargada de hacer un indicador a nuestro vertex y fragment shader, asignarle el archivo de texto del shader a cada indicador, compilar el código de shader, crear un objeto del programa y se atarle cada shader cuando se cargue el objeto del programa.
Nombre:	S_LoadTextFile
Descripción:	Función encargada de cargar el archivo texto para cada shader y lo retorna en un string.
Nombre:	S_GetVariable
Descripción:	Función encargada de devolver un id constante para una variable del shader.
Nombre:	S_SetInt
Descripción:	Función encargada de asignarle valores enteras a un id de variable del shader.
Nombre:	S_SetFloat
Descripción:	Función encargada de asignarle valores flotante a un id de variable del shader.

Tabla 5 Descripción de la clase CShader

Nombre: CTextura	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_texture	GLuint
m_textura	char*
m_cubemapid	GLuint
m_faceTarget	GLenum
Para cada responsabilidad:	
Nombre:	T_LoadBMP
Descripción:	Función encargada de cargar la imagen con la dirección pasada por parámetro.
Nombre:	T_LoadGLTexture
Descripción:	Función encargada de generar un id de la textura 2D que se crea devolviendo verdadero si se genero o falso si no se genera.
Nombre:	T_LoadGLCubeTexture
Descripción:	Función encargada de generar el id de la textura de tipo Cubemap que se crea al pasarle por parámetro seis imágenes de las cara del cubo y devuelve verdadero si se genera o falso si no se genera.

Tabla 6 Descripción de la clase CTextura

Nombre: CAplicacionOpengl	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_textura	CTextura*
m_shader	CShader*
m_boxsize	GLfloat
Para cada responsabilidad:	
Nombre:	AO_GenerarReflexionMap
Descripción:	Función encargada de generar el cubemap en modo de mapa de reflexión.
Nombre:	AO_DrawAgua
Descripción:	Función encargada de pintar el plano de agua en la escena.
Nombre:	AO_Inicializar
Descripción:	Función encargada de inicializar los parámetros a usar por la clase Controladora.
Nombre:	AO_CreateLake
Descripción:	Función encargada de renderizar la escena ha representar.
Nombre:	AO_ShaderManagement
Descripción:	Función encargada de gestionar el control del shader que se utilizara.

Tabla 7 Descripción de la clase CAplicacionOpengl

5.4 Diagramas de Secuencia

A continuación se presentan los diagramas de secuencia correspondiente a cada caso de uso planteado en el capítulo anterior. Con ellos se puede tener una idea del tiempo de ejecución de cada algoritmo que interviene en los CU.

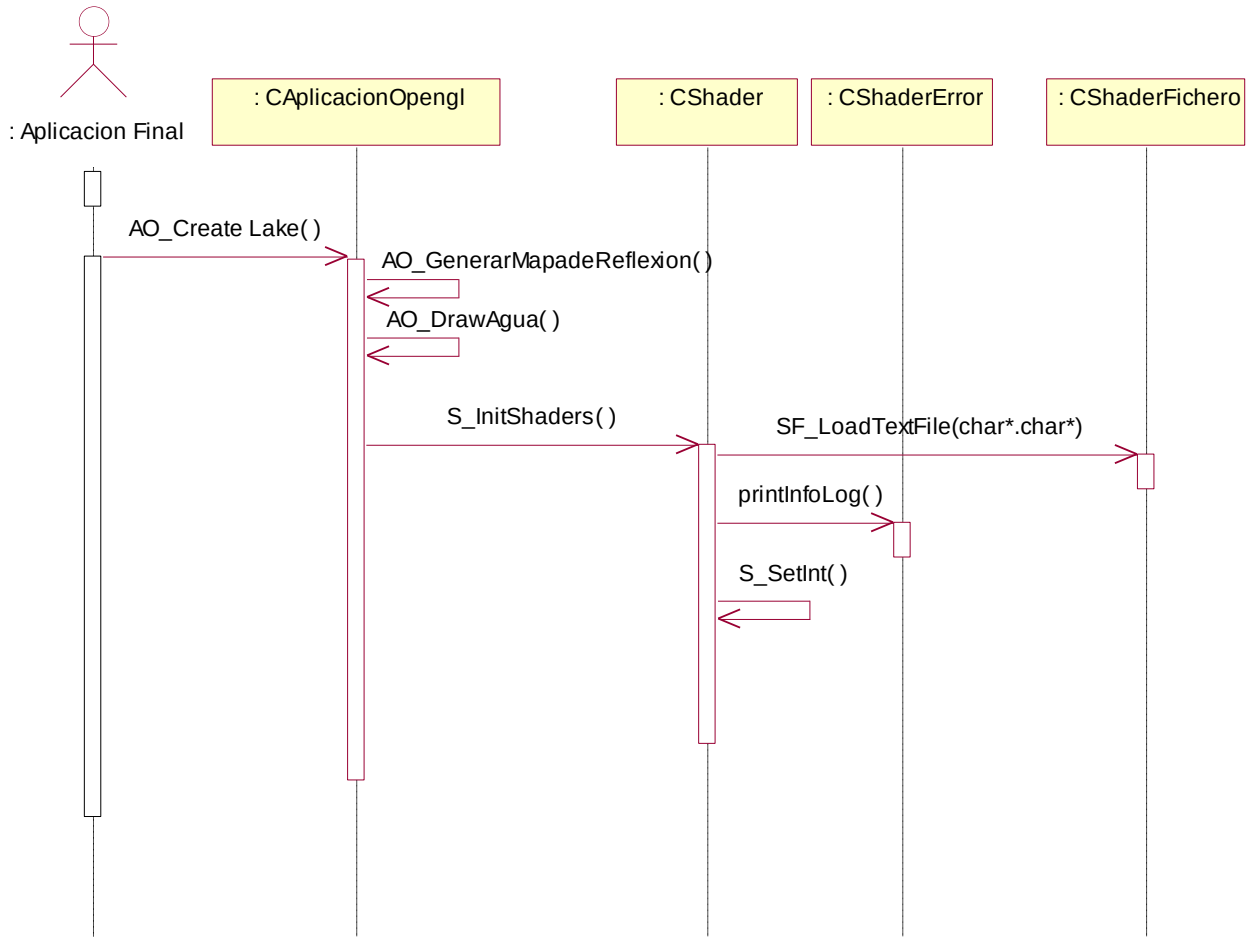

Figura 17 Diagrama de Secuencia del Caso de uso Crear Lago

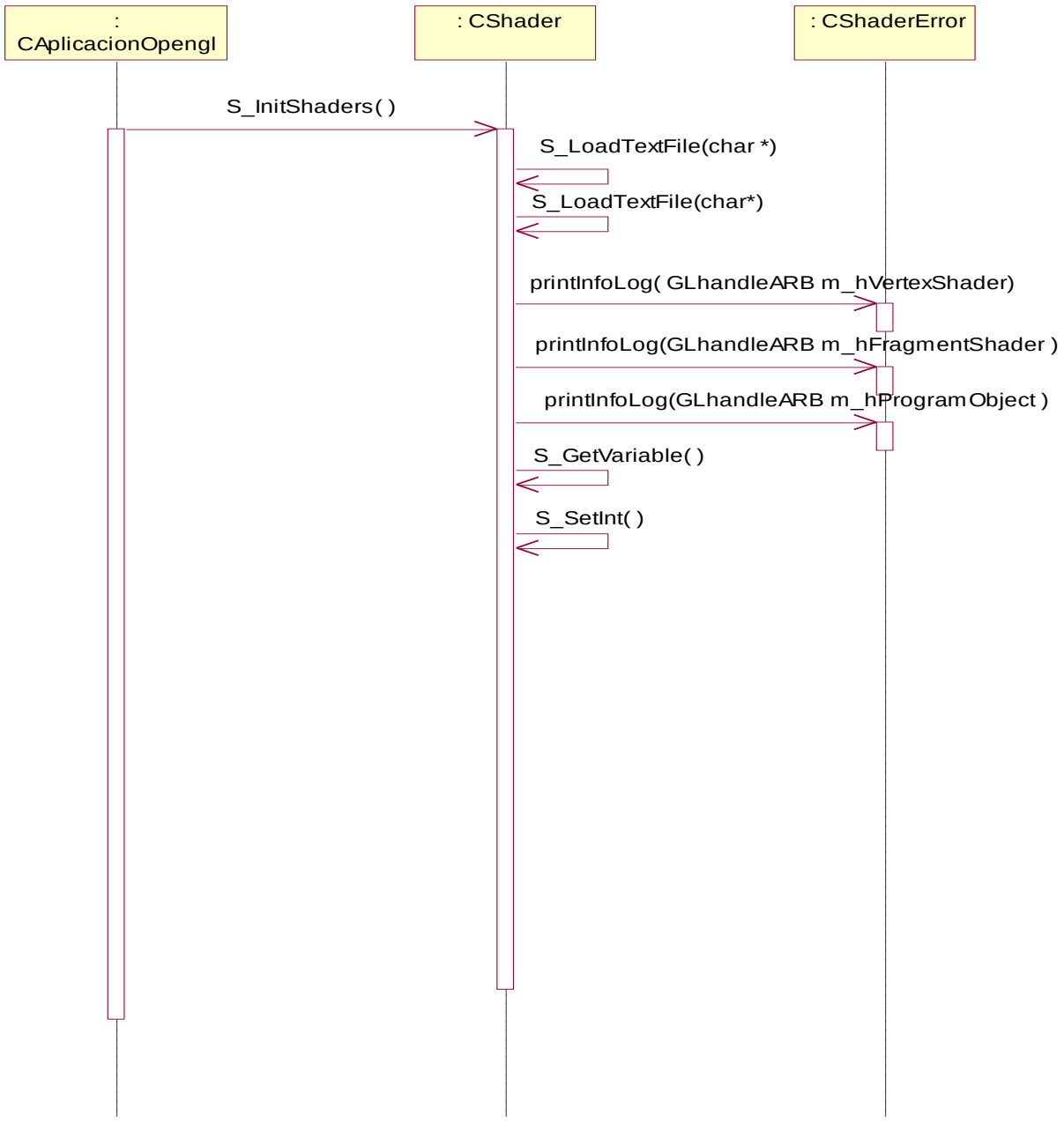

Figura 18 Diagrama de Secuencia del Caso de uso Gestionar Shader

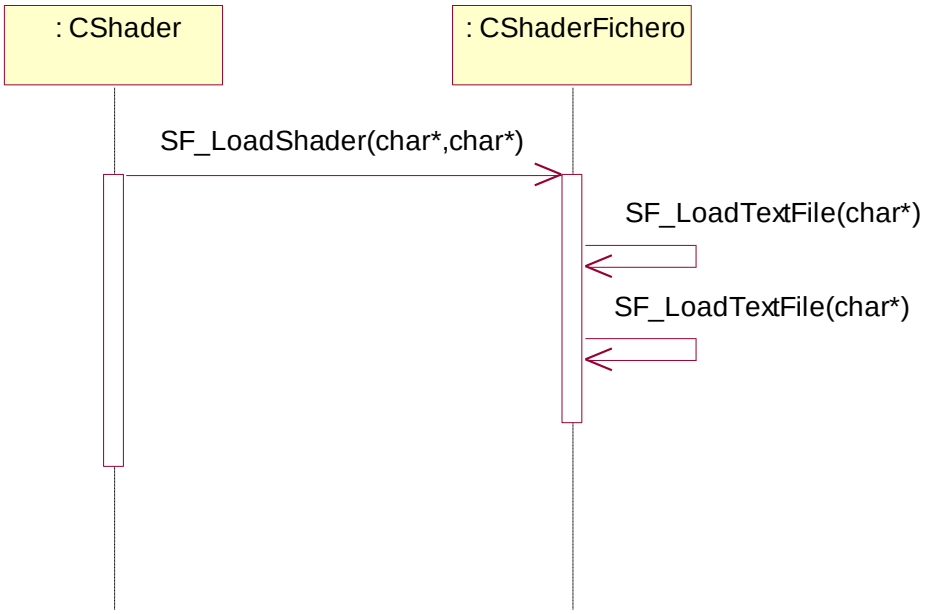

Figura 19 Diagrama de Secuencia del Caso de uso Cargar Shader

Capítulo 6 Implementación del sistema

Introducción

Esta etapa del proyecto constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .cpp correspondiente a la implementación en C++. Además se ejecuta la fase de prueba.

6.1 Estándares de codificación

Se programará en inglés, debido que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático. Se respetarán los estándares de codificación para C++ (identado, uso de espacios y líneas en blanco, etc.).

El conocimiento de los estándares seguidos para el desarrollo del módulo permitirá un mayor entendimiento del código, y es una exigencia de los autores de la misma que cualquier incorporación de efecto al deberá estar codificado siguiendo estos estándares.

Nombre de los ficheros:

Se nombrarán los ficheros .h y .cpp de la siguiente manera:

NameOfUnits.cpp

Se usará DW para identificar el nombre de la herramienta (en definición!!)

Constantes:

Las constantes se nombrarán con mayúsculas, utilizándose el "_" para separar las palabras: MY_CONST_ZERO = 0;

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Enumerados: enum **E**MyEnum {**ME**_VALUE, **ME**_OTHER_VALUE};

Indicando con “E” que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado. Véase otro ejemplo:

enum **E**Name**T**exture {**NT**_GEOMETRYNODE,…};

Estructuras: struct **S**MyStruct {…};

Indicando con “S” que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

Clases: class **C**ClassName;

Indicando con “C” que es una clase

Declaración de variables:

Los nombres de las variables serán escritas en minúscula. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador de la clase”m_” (en minúscula), si son globales se les antepondrá la letra “g”, y en caso de ser argumentos de algún método, se les precederá el sufijo “a”.

Tipos simples:

bool varname;

int name;

unsigned int **ui**Name;

float **f**Name;

char **c**Name;

char* **a**name; // arreglo de caracteres

char* **p**name; // puntero a un char

char** **aa**name; // bidimensional

char** **ap**name; // arreglo de punteros

bool **m**_membervarname; //variable miembro

char **g**globalvarname; //variable global, no se le antepone “”

short name;

Instancias de tipos creados:

EMyEnumerated name;

SMyStructure name;

CClassName objectname;

CClassName* **p**name; //puntero a objeto

CClassName* **a**name; //arreglo de objetos

CClassName* **a**name; // variable miembro de clase

Métodos:

En el caso de los métodos, empanzarán con mayúscula. Solamente los constructores y destructores comenzarán con “”.

En el caso de los argumentos se les precede con el sufijo “**a**”

Constructor y Destructor:

CClassName (bool varname**a**, float& varname**a**);

~**C**ClassName ();

Funciones:

bool Function1 (…);

int* Function_2 (…);

CClassName* Function3 (…);

Procedimientos:

void Procedure4 (…);

Métodos de acceso a miembros

Los métodos de acceso a los miembros de las clases se nombrarán “Gets” y “Sets”, pero con el nombre de la variable a la que se accede empezando con mayúscula y sin “m_”:

```
int myvar;  //variable
```

Obtención del valor:

```
int Get Myvar ();
{
   return myvar;
}
```

Establecimiento del valor:

```
void MyVar(char* myvara)
{
  myvar = myvara;
}
```

Obtención y establecimiento del valor:

```
int& Myvar ();

{

   return myvar;

}
```

6.2 Diagrama de componentes

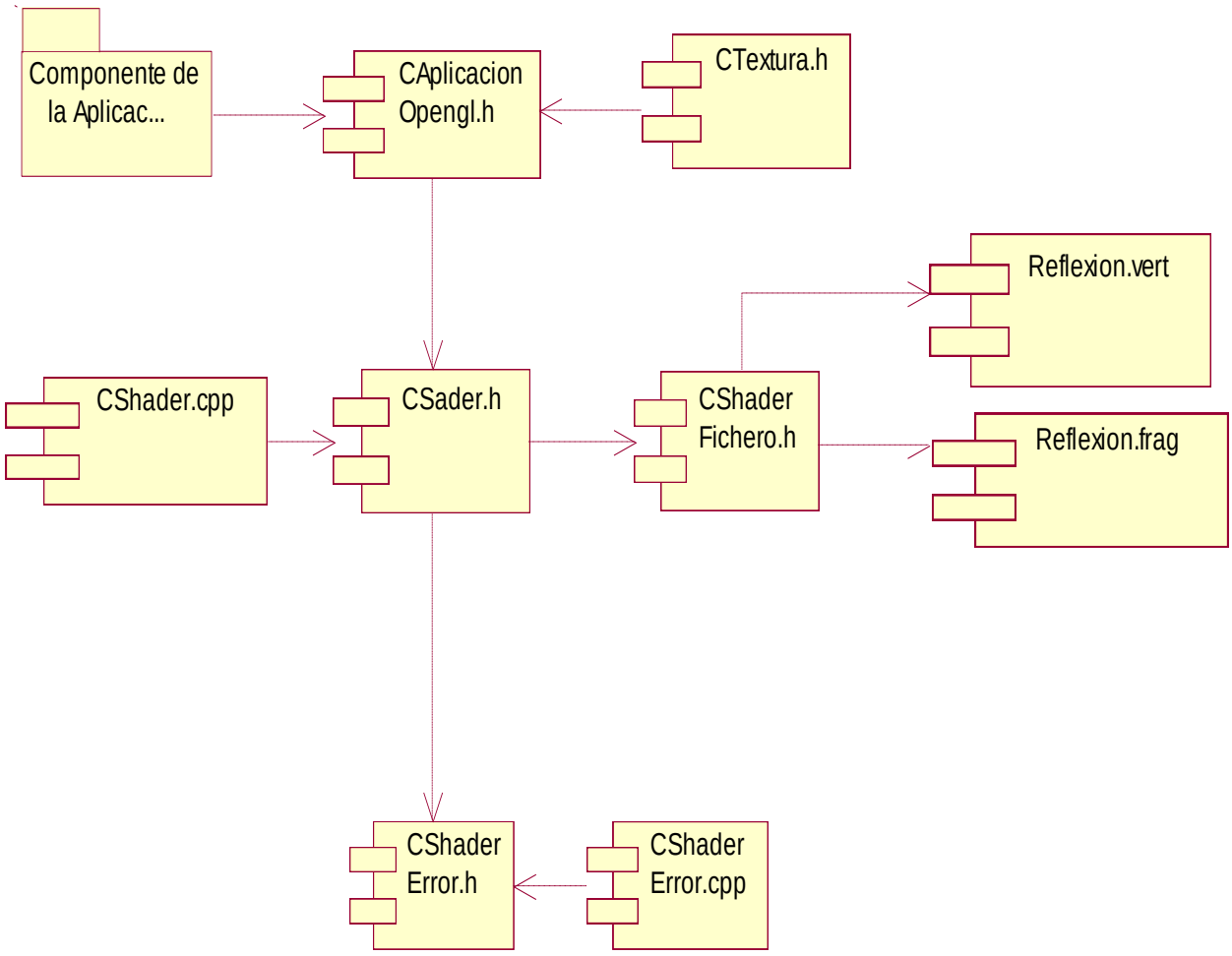

Figura 20 Diagrama de Componente

Conclusiones

Para dar cumplimientos a los objetivos de este proyecto, fue necesario primeramente hacer un riguroso estudio sobre las técnicas, tecnologías y tendencias en cuanto a la implementación de efectos visuales en los Sistemas de Entornos Virtuales, específicamente, lo relacionado con los efectos de deformación y el efecto de reflexión local y ambiental, dándosele implementación a esta última. Se analizaron los avances actuales en el hardware en cuanto al uso de las tarjetas gráficas y su máximo aprovechamiento para obtener efectos de alto realismo, los algoritmos más populares usados por las grandes compañías para la creación de entornos 3D, teniendo en cuenta las deficiencias de estos que debían ser erradicadas con este trabajo además de las posibilidades actuales de la Universidad de Ciencias Informáticas, especialmente la Facultad # 5 la cual se le asigno el Polo de Realidad Virtual del Centro . A partir de esta investigación, se propone una solución factible.

Después se realizó el proceso de Ingeniería del Software utilizando el Proceso Unificado del Software (RUP) como metodología de desarrollo, a partir del cual se obtuvo un módulo funcional que cumple con los requisitos de aplicaciones de este tipo como son modularidad, disponibilidad, etc. Además permite al programador cambiar la técnica de reflexión sin afectar la estructura del módulo, solo con un conocimiento mínimo de como funciona el mismo; además este está concebido para poder adaptarlo fácilmente a otra biblioteca gráfica, sin que sufra su estructura, así como a nuevas funcionalidades.

Recomendaciones

Se recomienda la extensión del módulo a la biblioteca gráfica *DirectX.*

Profundizar un poco más en las características y particularidades del lenguaje *HLSL* anteriormente expuesto.

Se recomienda además la incorporación del efecto visual de deformación de superficie del agua guiadas por el estudio que se hizo acerca de los algoritmos actuales más usados, utilizando la misma estructura propuesta para la representación del efecto visual de reflexión y refracción de entornos además que en su implementación se tenga en cuenta la interacción con objetos externos como por ejemplo embarcaciones, piedras, aire, etc.

Bibliografía

1. **Tessendorf, J.** *Simulating Ocean Water.* [En línea] 2001. [Citado el: 24 de enero de 2008.] http://home1.gte.net/tssnfrf/index.html.

2. *Luxpop.For Index of Refraction values and other photonic calculations and analysis.* [En línea] http://www.luxpop.com.

3. **Lasse Staff, Jensen y Golias, Robert.** Deep-Water Animation and Rendering. [En línea] 2001. http://www.gamasutra.com/gdce/2001/jensen/jenesen 01.html.

4. **Kaufman, Morgan.** Computer Animation: algorithms and techniques. [En línea] http://www.cis.ohio-state.edu/~parent/book/outline.html.

5. **Wales, Jimmy y Sanger, Larry.** GPU. [En línea] 2006. http://es.wikipedia.org/wiki/Graphics_Processing_Unit.

6. **J.Rost, Randi.** OpenGL® Shading Language, Second Edition,. [En línea] 2006.

7. **Claes, Johanson.** *Real-time water rendering.* s.l. : Lund University, March 2004. Master of Science thesis.

8. **Belyaev, Vladimir.** *Real-time simulation of water surface.* St. Petersburg, Russia : St. Petersburg State Polytechnical University, 2006.

9. **Foster, N y Metaxas, D.** *"Realistic Animation of Liquids" Proceeding GI.* 1996. págs. 204-212.

10. **Fournier, A y Reeves, W.T.** *"A Simple Model of Ocean Waves".* 1986. págs. 75-84. Vol. 20.

11. **Kass, M y Miller, G.** *"Rapid,stable fluid dynamic for computer graphic".* 1990. págs. 49-57. Vol. 24.

12. **Willern H, de Boer.** *Fast Terrain Rendering Using Geometrical Mipmapping.* 2000.

Referencias Bibliográficas

1. **Tessendorf, J.** *Simulating Ocean Water.* [En línea] 2001. [Citado el: 24 de enero de 2008.] http://home1.gte.net/tssnfrf/index.html.

2. *Luxpop.For Index of Refraction values and other photonic calculations and analysis.* [En línea] http://www.luxpop.com.

3. **Lasse Staff, Jensen y Golias, Robert.** Deep-Water Animation and Rendering. [En línea] 2001. http://www.gamasutra.com/gdce/2001/jensen/jenesen 01.html.

4. **Kaufman, Morgan.** Computer Animation: algorithms and techniques. [En línea] http://www.cis.ohio-state.edu/~parent/book/outline.html.

5. **Wales, Jimmy y Sanger, Larry.** GPU. [En línea] 2006. http://es.wikipedia.org/wiki/Graphics_Processing_Unit.

6. **J.Rost, Randi.** OpenGL® Shading Language, Second Edition,. [En línea] 2006.

7. **Claes, Johanson.** *Real-time water rendering.* s.l. : Lund University, March 2004. Master of Science thesis.

8. **Belyaev, Vladimir.** *Real-time simulation of water surface.* St. Petersburg, Russia : St. Petersburg State Polytechnical University, 2006.

9. **Foster, N y Metaxas, D.** *"Realistic Animation of Liquids" Proceeding GI.* 1996. págs. 204-212.

10. **Fournier, A y Reeves, W.T.** *"A Simple Model of Ocean Waves".* 1986. págs. 75-84. Vol. 20.

11. **Kass, M y Miller, G.** *"Rapid,stable fluid dynamic for computer graphic".* 1990. págs. 49-57. Vol. 24.

12. **Willern H, de Boer.** *Fast Terrain Rendering Using Geometrical Mipmapping.* 2000.

Índice de figuras, ecuaciones y tablas

Índice de figuras

Índice de Ecuaciones

Índice de Tablas

Glosario de Abreviaturas

Dinámica de Fluido en la Computación (CFD) la ecuación de Navier-Stoke (NSE)

2D: Dos dimensiones.

3D: Tres dimensiones.

API: Application Programmer's Interface, (interfaces para programadores de aplicaciones).

CPU: Unidad Central de Procesamiento.

CU: Caso de Uso.

CFD: Dinámica de Fluido en la Computación

FT: Transformaciones de Fourier

FFT: Rápidas transformaciones de Fourier.

GLSL: Opengl Shading Language. Lenguaje de programación de alto nivel para realizar aplicaciones gráficas y lograr mayor eficiencia y realismo.

GPU: Unidad de Procesamiento Gráfico.

HLSL: *High Level Shading Language*. Lenguaje de programación de alto nivel para realizar aplicaciones gráficas y lograr mayor eficiencia y realismo.

HW: *Hardware*.

RV: Realidad Virtual.

NSE: Ecuación de Navier-Stoke

Glosario de Términos

A:

Aliasing: Las líneas, especialmente las que están casi horizontales o verticales, aparecen dentadas o irregulares debido a su representación por *píxels*. Este escalonamiento es llamado *aliasing*.

*****

D:

Dinámica: Calificativo que sugiere la actividad, el movimiento, el cambio, la transformación estructural y funcional.

*****

E:

Estado de Shader: Información de estados de iluminación y sombras para representar en las escena.

Estructura: Conjunto de propiedades y funciones que definen en elemento.

*****

H:

Hardware gráfico: Dispositivos necesarios para crear aplicaciones gráficas.

*****

R:

Ray-tracing: Trazado de rayo

Reverse ray-tracing: Inverso trazado de rayo

*****

M:

Malla: Forma de representar un modelo a partir de polígonos. Colección de vértices, aristas y polígonos conectados de forma que cada arista es compartida como máximo por dos polígonos.

Modelo: Prototipo para la animación.

*****

N:

Normal: (ver vector normal).

Normalizar: Transformar coordenadas o parámetros que tengan un valor predeterminado.

*****

P:

Perturbación: Evento que altera o modifica los acontecimientos normales.

Pipeline gráfico: Conjunto de procedimientos para lograr obtener una aplicación gráfica.

Píxel: Abreviatura de “picture element”. Es la menor unidad de información de una imagen digital.

Píxel shader: Encargado de realizar todas las operaciones a nivel de *píxel* como es el cálculo de la iluminación *per-píxel*, mapeo de texturas, uso de los mapas de normales para la determinación de la normal del píxel, etc.

*****

R:

Realidad Virtual: Simulación de un medio ambiente real o imaginario que se puede experimentar visualmente en tres dimensiones. La realidad virtual puede además proporcionar una experiencia interactiva de percepción táctil, sonora y de movimiento.

Recursos: Son los elementos de una computadora que utilizan los dispositivos para poder funcionar correctamente.

Reflexión: Fenómeno que ocurre cuando la luz incide sobre una superficie y es desviada por ésta sin cambiar de medio.

Reflexión Especular: La reflexión es especular cuando la superficie es lisa, y difusa cuando la superficie es rugosa.

Rendereado (rendering): Crear en forma automática una imagen de acuerdo al modelo tridimensional que existe en el ordenador.

*****

S:

Shader: Un *shader* define las características finales de un objeto. Por ejemplo, un *shader* puede definir el color y la reflectividad de una superficie.

Sistema de funciones fijas (Fixed-Function): Es uno de los dos métodos que se utilizan actualmente para modificar los resultados gráficos; el otro es el Sistema de Funciones Programables, también conocido por *Shader*.

*****

T:

Tarjeta gráfica: Es una tarjeta de circuito impreso encargada de transformar las señales eléctricas que llegan desde el microprocesador en información comprensible y representable por la pantalla del ordenador.

Textura: Imagen que sirve de “piel” a los modelos en un mundo virtual.

*****

V:

Vector: Cantidad que expresa magnitud y dirección.

Vector normal: Vector cuyos puntos están en dirección perpendicular a una superficie.

Vertex shader: Encargados de transformar todos los vértices de la escena. En ellos se ejecutan las transformaciones de espacio objeto a espacio de mundo, de cámara, y finalmente se obtiene la posición en la pantalla.

*****