Universidad de las Ciencias Informáticas.
Facultad#5

Efectos Especiales para Entornos de Realidad Virtual

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Jaime González Campistruz

Lannie Octavio Herrera Pérez

Tutor: Fernando Jiménez López

Asesoría: MSc. Pedro Carlos Pérez Martinto

Ciudad de la Habana, Julio de 2007

"Año 49 de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los 3 días del mes de _Julio_ del _*2007*__.

_____	_____
Firma del Autor	Firma del Autor
(Jaime González Campistruz)	(Lannie Octavio Herrera Pérez)

_____

Firma del Tutor

(Lic. Fernando Jiménez López)

Datos de Contacto

Nombre y Apellidos: Fernando Jiménez López

Edad: 26 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Informática

Categoría Docente: Profesor Instructor

E-mail: fjimenez@uci.cu

Graduado de la CUJAE, con tres años de experiencia en el tema de la Gráfica Computacional, y líder de un proyecto de Realidad Virtual en la Universidad de las Ciencias Informáticas.

Agradecimiento

A todas las personas que nos han apoyado en la realización de este proyecto, a nuestro tutor Fernando, y en especial a la Universidad de las Ciencias Informáticas por hacer realidad nuestro sueño.

Dedicatoria

…este trabajo de diploma está dedicado a mi familia y en especial a mi linda hermana Lissette, a mis padres Octavio y Amparo por dejarme cumplir este sueño.

…a mi novia Dayra por su atención y comprensión en todos los momento vividos.

Lannie

…mis eternos agradecimiento a todos aquellos que hicieron posible que cumpliera este sueño; a mis compañeros y grandes amigos, a mi querida novia Yohana, y en especial, a mis padres Margarita y Nelson y a mis hermanos Nelson y Javier que han estado conmigo en todos los momentos buenos y malos de mi vida.

Jaime

Resumen

En la actualidad los Sistemas de Realidad Virtual (SRV) y más específico los efectos de pirotecnia han tenido un gran impulso en el mundo de los ordenadores, ya sea porque se busca una mayor calidad visual en las simulaciones o para disminuir el rendimiento del coste computacional en esta rama de la industria informática.

Este trabajo se basa en realizar un subsistema para la creación de efectos de pirotecnia como son humo, explosiones y fuego fundamentalmente, teniendo en cuenta para esto el incremento de la calidad visual y lograr una optimización del sistema en toda su magnitud. Para alcanzar esta meta, se estudian métodos de visualización lo que permitirá llegar a la conclusión cual se escogerá para el proyecto, y se trabaja en la investigación y desarrollo de algoritmos eficientes para la representación tridimensional en tiempo real. El subsistema resultante de esta investigación, permitirá un mayor acercamiento al mundo de los efectos de pirotecnia con menos coste computacional.

Palabras claves:

Módulo, partículas, efecto, pirotecnia, diagramas.

Tabla de contenido

Introducción

Desde años anteriores la humanidad a seguido muy de cerca la investigación para el desarrollo de la Realidad Virtual (RV) y sus entornos virtuales, obteniendo grandes aportes con el transcurso de los años siendo mayor su desarrollo y mayor su aplicación. De esta forma la realidad virtual entra en un exclusivo rango de herramientas para hacer; como son los juegos y los simuladores. En el cual el usuario puede incursionar creativamente, hasta donde el límite de su imaginación se lo permita.

En la industria de los videos juegos y de los simuladores, la modelación y simulación de efectos especiales, como explosiones, fuego, destellos, humo, lluvia y otros; tienen una gran importancia debido a que estos incrementan enormemente la calidad visual del entorno virtual que se simula, incluso, permite al usuario de la aplicación introducirse dentro de la simulación, el cual es uno de los objetivos principales de la RV.

En Cuba, una de las empresas que ha impulsado el desarrollo de los Sistemas de Realidad Virtual (SRV), es el “Centro de Investigación y Desarrollo #2” (CID2), conocido en el mercado como **SIMPRO** (Simuladores Profesionales). Esta empresa comenzó como proyecto desde el año 1994, y oficialmente existe con el nombre de SIMPRO desde el año 2000; la cual en vínculo con la Universidad de Las Ciencias Informáticas (UCI) ha desarrollado una herramienta de visualización con diferentes subsistemas incluidos como el de sonido, el de inteligencia artificial y otros necesarios para crear los diferentes simuladores que esta desarrolla, pero no posee un subsistema que permita la visualización de estos efectos especiales.

Hasta ahora con la herramienta solo se ha hecho explosiones muy rudimentarias de tipo 2D, las cuales poseen poca calidad visual y poco realismo, pero nada referente a explosiones lo suficientemente reales, humo, fuego, chispas y entre otros efectos de la pirotecnia, esta parte está completamente desde cero. Esto conlleva a que el proyecto se limite a no usar estos efectos visuales en sus aplicaciones, reduciendo su posibilidad de emerger en el mercado mundial debido a la falta de realidad y calidad del software.

Analizando la situación existente con la herramienta, se propone como **problema científico** a resolver con este trabajo, ¿Cómo crear un módulo de efectos de pirotecnia empleando los fluidos estables?

Este trabajo tiene como **objeto de estudio** los procesos de generación de efectos especiales para sistemas de realidad virtual, y se tendrá como **objetivo de investigación** el siguiente:

1. Concebir un subsistema lo suficientemente óptimo y extensible de generación de efectos especiales de pirotecnia en tiempo real.

Esto se hará enfocándose en el siguiente **campo de acción**, los efectos especiales de pirotecnia para sistemas de realidad virtual.

Para el cumplimiento de los objetivos planteados anteriormente se trazan las siguientes **tareas** a desarrollar:

1. Estudiar los métodos de visualización existentes.
2. Estudiar los métodos físicos.
3. Seleccionar las herramientas a utilizar.
4. Desarrollar las soluciones técnicas para alcanzar los objetivos propuestos.
5. Implementar una primera versión del módulo, desarrollando el efecto de fuego.
6. Realizar un diagrama de clases lo suficientemente extensible que permita la inclusión de nuevos efectos de pirotecnia.

Capítulo 1: Fundamentación Teórica

Introducción

Lograr optimizar la simulación de los diferentes tipos de efectos especiales usados en la industria de los videos juegos, el cine y los simuladores es cosa que hoy en día todavía se investiga con profundidad. Mantener e incrementar un sistema óptimo y lo suficientemente real ha sido y será el principal objetivo de los investigadores desde años atrás pues es y será una demanda presente en el campo de la realidad virtual y de los efectos especiales en la industria de los videos juegos. Con la rápida evolución de las tarjetas gráficas, lograr efectos en tiempo real es ahora posible, cosa que fue imposible años atrás, pero todavía existe una continuidad y deseos de incrementar el realismo obtenido. La visualización de fenómenos naturales puede ser usada para incrementar el realismo de los entornos virtuales, convenciendo al usuario de suspender sus dudas y de emergerse más en la simulación.

Los efectos de pirotecnia como fuego, humo y explosiones son los más usados en el mundo de la RV, y son los más difíciles de simular, empleando los tradicionales métodos de visualización que son basados o no en métodos físicos. Como la explosión y el fuego son fenómenos muy intensos, estos deben ser reproducidos de forma real en orden de convencer el ojo humano.

Este capítulo profundizará en los diferentes métodos de simulación y visualización existentes en el mundo para la creación de estos efectos, así como también conceptos generales de las leyes físicas que rigen a estos fenómenos. Para lograr esto, se ha divido el proceso de simulación en dos partes, métodos de simulación física y métodos de visualización del efecto, con el objetivo de dar mayor claridad al texto y de dejar claro que diferentes métodos de simulación y visualización pueden ser combinados a gusto del desarrollador.

1.1 Conceptos previos de los fenómenos o efectos de pirotecnia

Para lograr tener una idea de cómo simular los fenómenos naturales, llamados efectos especiales en el campo de la realidad virtual, que en el caso de esta tesis sería los efectos de explosiones y fuego; es necesario tener bien claro cuáles son las leyes y los conceptos que los rigen. En este epígrafe se dará una breve descripción de ello, tanto del fuego como de las explosiones.

1.1.1 Teoría del fuego

El fuego es un rápido y persistente cambio químico que libera calor y luz y es acompañado por llamas, es especialmente la oxidación exotérmica de la sustancia combustible (1).

Existen tres aspectos centrales del fuego, el aspecto químico de la combustión, el aspecto físico de la transferencia de calor y al aspecto mecánico de la transferencia de masa.

1. La combustión envuelve a la reacción química, donde los principales reactivos son el oxidante y el vapor de gas del combustible.
2. La transferencia de calor consiste en la difusión, convección y radiación. Difusión y convección son principalmente responsables del esparcimiento del fuego, mientras que la radiación de cuerpo-negro releva la apariencia visual que se asocia como el fuego.
3. La transferencia de masa debido al flujo y los efectos limitantes, es responsable de las características de forma y movimiento de la llama del fuego. Otro aspecto de la trasferencia de masa es la difusión molecular debido a las diferencias en la composición del gas en varias partes de la llama del fuego.

Combustión: La combustión es una reacción de oxidación-reducción, donde el agente oxidante es el oxígeno y el agente reductor es el combustible. El combustible que no se encuentre en su estado gaseoso, necesita vaporizarse antes de que pueda reaccionar con el oxidante y realizar la combustión. La vaporización ocurre cuando el combustible está suficientemente caliente, transformando el combustible líquido o

sólido en gas combustible. En adición, el gas combustible necesita alcanzar su temperatura de ignición para que la combustión tenga efecto. La ecuación general para la combustión es la siguiente:

$$\alpha \text{ oxidante} + \beta \text{ combustible} \rightarrow \varepsilon\ CO_2 + \delta\ H_2O + \ldots$$

La parte derecha de la ecuación muestra el oxidante y el combustible, los cuales son prerrequisitos para la combustión. Durante la combustión son formados CO_2, vapor de agua (H_2O), gases calientes y productos como hollín de carbón. También es producido más calor, lo cual provoca más combustión, manteniendo el proceso activo hasta que no exista más combustible, calor u oxígeno (1).

Cuando ocurre la combustión pueden aparecer dos tipos de llamas; llamas de difusión y llamas pre mixtas. Ejemplo de difusión son las llamas de los candiles y de las segundas son las llamas azuladas vistas en quemadores de gas natural.

Radiación de cuerpo-negro: Es emitida durante la reacción de combustión por los productos gaseosos, en particular el hollín del carbón, y es la responsable de dar la característica anaranjada amarillosa que tiene el color del fuego (2).

Durante la reacción, es generada una buena cantidad de calor, trayendo consigo más combustión y la creación de productos de gases. La temperatura decrecerá a medida que se va alejando de la zona de reacción, por lo que la radiación de caja negra disminuirá también hasta que el color anaranjado amarilloso no sea visible. Cuando los productos de los gases sean lo suficientemente coloreados, entonces aparecerá el hollín de humo.

Cuando nos referimos a los cuerpos negros emitiendo radiación no es más que partículas de hollín que son creadas antes y durante la combustión. Para el cálculo de esta intensidad se utiliza la fórmula de Planck, la cual da la intensidad de cierta longitud de onda λ irradiada por el cuerpo-negro con una temperatura *T*, donde *h* es la constante de Planck, *c* es la velocidad de la luz y *k* es la constante de Boltzamann (1).

$$B_\lambda(T) = \frac{2\pi hc^2}{\lambda^5\left(e^{\frac{hc}{\lambda kT}} - 1\right)}$$

1.1.2 Teoría de las explosiones

La definición exacta de una explosión varía dependiendo del contexto, pero generalmente una explosión es una liberación repentina de energía que crea un frente de propagación de presión denominada onda de choque, que se desplaza alejándose de la fuente mientras va disipando energía. Las explosiones pueden provenir de un evento mecánico, químico o nuclear (5).

La onda de choque es el principal efecto de la explosión, se mueve a velocidades supersónicas y es solamente visible manifestando una delicada refracción de luz. Excepto para explosiones a grandes escalas, la refracción de luz es completamente invisible para los ojos humanos.

Los efectos secundarios de la explosión incluyen destellos brillantes, fuego, polvo y escombros voladores. Estos si son claramente visibles.

Existen dos tipos de explosiones:

Explosiones físicas: son aquellas en las que hay una liberación de gas que está sujeta a presión sin que haya una transformación química del gas. Por ejemplo una compresora que genera aire, por una falla de la misma puede originar una explosión del recipiente que lo contenga, pero no hay una transformación de aire, sólo es una liberación del mismo.

Explosiones químicas: son aquellas en la que existe una transformación de las sustancias. Es decir, hay una transformación química. En estas hay explosiones generadas por gas licuado de petróleo (LP), gas natural, vapores de líquidos inflamables como gasolina, por polvos inflamables dispersos en el medio ambiente, y por explosivos.

Dentro de las explosiones químicas se encuentran las siguientes:

Explosiones de partículas suspendidas, que ocurren cuando una sustancia altamente inflamable como la pólvora es dispersada en el aire y después es incendiada. Posibles mecanismos de dispersión de las partículas podrían ser vibraciones o pequeñas explosiones iniciales.

Explosiones de vapor líquido, que no son más que aquellas que ocurren cuando un líquido inflamable y su vapor, son dispersados en el escenario y después es incendiado.

El proceso de reacción química que ocurre en la explosión, se basa en la mencionada en el epígrafe anterior, referente a los aspectos de fuego. De hecho, se puede decir que una explosión es un tipo de fuego. Debido a que el comportamiento de la

explosión está regido por su transferencia de calor, su combustión y su transferencia de masa.

1.2 Los Fluidos

Las principales características que tienen en común los efectos de pirotecnia y otros efectos en general, es que estos se rigen en parte por la dinámica de los fluidos, dinámica que se ha venido estudiando desde hace años por científicos en la materia, y que podemos generalizar para su uso. Por lo que juega un papel crucial saber sus conceptos generales.

Un **fluido** es una sustancia que fluye bajo presión y la mecánica de fluido es la parte de la dinámica de fluidos encargada del estudio del movimiento de los fluidos.

El flujo de los fluidos puede ser caracterizado por una serie de aspectos que gobiernan el comportamiento del flujo (2):

1. Flujos Comprensibles, como opuesto de los flujos incomprensibles, ocurren cuando la variación de la presión es lo suficientemente grande como para incurrir cambios substanciales en la densidad.
2. Flujos Viscosos, como opuesto de los flujos no viscos, es usado para modelar fluidos en los cuales la fricción interna tiene un efecto significante. La cantidad de fricción es descrita por la viscosidad del fluido.
3. Flujos estables, como opuesto de los inestables, es usado cuando el tiempo de cambio de las propiedades del fluido es cero, es decir, cuando las propiedades del fluido se mantienen constantes.
4. Flujos Turbulentos, son aparentemente caóticos y son causados por los inestables vórtices que aparecen en muchas escalas e interactúan entre ellos. Los fluidos con carencia de turbulencia son llamados laminares y son aparentemente suavizados y estables.

1.2.1 Computación de la dinámica de los fluidos (CFD)

A la rama que estudia los fluidos de manera computacional se le conoce como CFD. Es la aplicación de computadoras para analizar o resolver problemas en la dinámica de los fluidos, que gracias al exponencial incremento de la potencia del hardware, se ha aumentado en experiencia y resultado. Su aplicación varía entre investigaciones

físicas complejas y simulaciones en la industria de efectos especiales, tanto para juegos como para películas.

Para desarrollar su simulación, existen diferentes algoritmos, pero el más común de todos es discretizar el dominio del espacio de simulación en una maya de pequeñas celdas. Cada celda contiene el estado del fluido en la parte del dominio que le corresponde, que usualmente son estados como velocidad, densidad, temperatura y presión. Estos estados son después actualizados con un paso de tiempo discreto, con el objetivo de estudiar como el fluido se desarrolla al paso del tiempo. La simulación gobierna la evolución del campo de velocidad del fluido en el dominio de desarrollo.

El paso de actualizar las celdas, es típicamente hecho resolviendo las ecuaciones de Navier-Stokes, que son capaces de modelar flujos comprensibles, viscosos, inestables y turbulentos. Los datos obtenidos pueden ser visualizados o analizados dependiendo del objetivo con el cual fue desarrollado.

1.2.2 Ecuaciones de Navier-Stokes para un flujo incomprensible

El flujo de fluidos es comúnmente modelado usando campos de vectores. Las ecuaciones de Navier-Stokes son las ecuaciones de derivadas parciales fundamentales que describen el comportamiento de fluidos incompresibles a través de un campo de velocidad; y que han sido objeto de uso desde sus inicios en el campo de CFD (4). Tales ecuaciones son:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla \mathrm{p} + \nu \nabla^2 \mathbf{u} + \mathbf{F}$$

$$\nabla \cdot \mathbf{u} = 0$$

Donde **u** es el campo de velocidad del fluido y la primera ecuación da la derivada parcial de **u** con respecto al tiempo, ρ es la densidad (constante) del fluido, ν es el coeficiente de viscosidad y $\mathbf{F} = (fx, fy)$. 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.

Los términos de la parte derecha de la primera ecuación son los de advection[1], presión, difusión y de fuerza respectivamente y son aceleraciones. Términos que describen las principales características de los fluidos (3).

Advection: No es más que el movimiento propio del fluido, la velocidad del fluido causa la trasportación de objetos, densidades y otras cantidades junto con el fluido, lo cual hace que si se derrama tinta sobre un fluido en movimiento, esta siga el movimiento del fluido. En las bibliografías existentes se le conoce como *self-advection*[1] o *término de advection*[1] y en las ecuaciones de Navier-Stokes es el primer término de la parte derecha de la primera ecuación.

Presión: Cuando una fuerza es aplicada al fluido, esta no se propaga instantáneamente por todo el fluido. En lugar de esto, lo que ocurre es que las moléculas cercanas al lugar donde se emitió la fuerza empujan aquellas moléculas lejanas al lugar. Debido que la presión es la fuerza por unidades de área, cualquier presión en el fluido es una aceleración. En la primera ecuación, es el segundo término de la parte derecha.

Difusión: Es el término que determina la densidad de un fluido, algunos fluidos son más densos o espesos que otros. Se dice que un fluido denso tiene una alta viscosidad, que significa cuan resistente es el fluido al flujo. Esta resistencia es resultado de la difusión del impulso y por lo tanto, velocidad. El tercer término de la ecuación es el llamado término de difusión.

Fuerzas externas: Los cuatro términos encapsulan aceleración debido a las fuerzas externas aplicadas al fluido. Estas fuerzas pueden ser fuerzas locales o fuerzas de cuerpo. Las fuerzas locales son aquellas que actúan en una específica región. Las fuerzas de cuerpo, como la fuerza de gravedad, se aplican en el fluido entero.

En el transcurso del tiempo muchas soluciones numéricas se han planteado para solucionar estas ecuaciones, pero la mayoría de estas requieren de mucho tiempo de procesador para solucionarla, y esto es debido a que tradicionalmente el trabajo con la CFD ha sido más enfocado a la precisión que a la eficiencia, porque su aplicación a sido más en los campos de la ingeniería y la industria que para la realidad virtual, siendo el factor gobernante la exactitud numérica y no la calidad visual.

Los modelos clásicos son inestables, lo que significa que el rendimiento de la simulación se deteriora rápidamente o diverge cuando el paso de tiempo se torna muy grande, lo que los hace imposibles de usar en aplicaciones de tiempo real.

1.2.3 Método estable de Jos Stam

Un método estable de resolver las ecuaciones de Navier-Stokes fue dada por Jos Stam en su trabajo ¨Stable Fluids¨ donde se hace referencia en (3), para resolver fluidos incomprensibles y homogéneos, es decir, fluidos cuya densidad se mantiene constante en el espacio y tiempo. Stam sacrifica exactitud numérica por calidad visual y rapidez. Su método es muy estable y nunca explota y puede ser usado con grandes pasos de tiempo. Esto es un prerrequisito crucial para la creación de efectos visuales o especiales en tiempo real.

Stam simuló la dinámica del fluido en una maya cartesiana con coordenadas espaciales $\mathbf{x} = (x, y)$ y con una variable de tiempo *t*. Representó el campo de velocidad como $\mathbf{u}(\mathbf{x}, t)$ y el campo escalar de presión como $p(\mathbf{x}, t)$.

Las ecuaciones son aproximadas, con el objetivo de establecer el cálculo del campo de velocidad sobre un simple paso de tiempo Δt.

Primer Paso:

Primeramente el término de las fuerzas externas es adicionado al campo de velocidad mediante la siguiente ecuación:

$$\hat{\mathbf{u}}(x) = \mathbf{u}(x) + \mathbf{f}(\mathbf{x})\Delta\mathbf{t}$$

Donde el término de la izquierda es el campo de velocidad después de agregado el término de fuerza. Las fuerzas externas son aplicadas al fluido de manera manual.

Segundo Paso:

Después, el campo de velocidad se actualiza con su movimiento propio, (advected[1]), lo que puede interpretarse como la velocidad empujándose ella misma hacia adelante. Stam usa la siguiente ecuación, que tiene una funcionalidad implícita que permite un mejor cálculo de la operación:

$$\hat{\mathbf{u}}(x) = \mathbf{u}(\mathrm{x} - \mathbf{u}(\mathrm{x})\Delta\mathrm{t}).$$

Una forma más intuitiva de ver esta ecuación sería ver el campo de velocidad como un conjunto de partículas centradas en cada celda de la maya. Cada partícula es trazada hacia atrás en un paso de tiempo usando su velocidad actual, para encontrar la posición previa que la partícula debe haber tenido para terminar en la posición actual. La velocidad en la posición previa es entonces interpolada entre las celdas vecinas

para obtener la nueva velocidad actual en la posición actual. La fig. 1 muestra claramente este proceso.

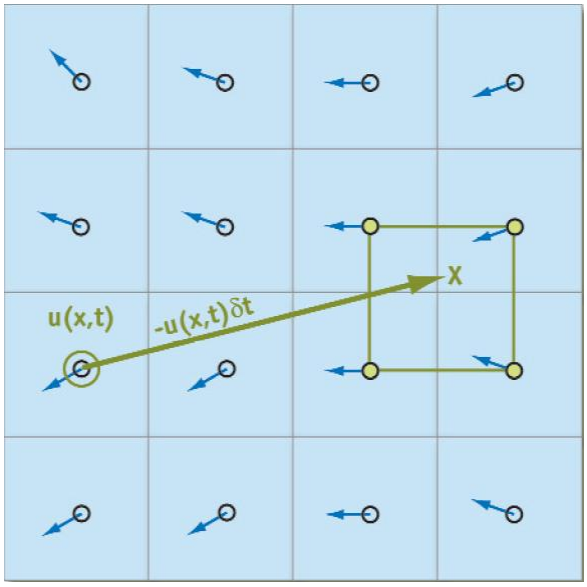

Figura 1 Proceso de interpolación

Tercer Paso:

Después de que el término de advetion[1] ha sido evaluado, el término de difusión $\nu\nabla^2\mathbf{u}$ es adicionado. La viscosidad describe la resistencia del fluido al moverse, causado por la fricción existente dentro de él. Igual que en el paso anterior, en la difusión también se aplica la misma funcionalidad:

$$(\mathbf{I} - \nu\Delta t\nabla^2)\hat{\mathbf{u}}(x) = \mathbf{u}(x).$$

Donde $\mathbf{I}$ es la matriz de identidad. La ecuación es usada para encontrar el nuevo campo de velocidad û. Esta ecuación encapsula un sistema de ecuaciones lineales que pueden ser resueltos iterativamente con los métodos de Gauss-Seidel o el método de Jacobi.

Cuarto Paso:

Después que los términos de fuerza, advection[1] y de difusión hayan sido adicionados, el campo de velocidad que se obtiene es divergente, por lo que la conservación de la masa no será válido:

$$\nabla \cdot \mathbf{u} \neq 0$$

Sin embargo, según el teorema de descomposición "*Helmholtz-Hodge*"; cualquier campo de vectores puede ser descompuesto en la suma de sus componentes, un no divergente campo vectorial y el gradiente de un campo escalar. El campo escalar correspondiente a la descomposición del campo de velocidad es el término de presión *p.* Calculando y sustrayendo el gradiente de *p* al campo de velocidad u, se obtiene como resultado el campo no divergente de velocidad, como se muestra en la siguiente ecuación:

$$\mathbf{u} = \hat{\mathbf{u}} + \nabla p.$$

Donde û es el campo no divergente de velocidad. El término de la presión *p* es calculado resolviendo la ecuación de Poisson resultante de la multiplicación de cada lado de la ecuación anterior por "∇":

$$\nabla \cdot \mathbf{u} = \nabla^2 p.$$

Nota: En esta ecuación el término $\nabla \cdot \hat{u} = 0$ debido a que es un campo no divergente.

Igual que en la difusión, la ecuación encapsula un sistema de ecuaciones lineales. Stam tiene en cuenta cuando va a resolver la ecuación de Poisson para la difusión; las condiciones de límite de Neumann, con el objetivo de definir las condiciones de límite del campo escalar del fluido, *p,* asegurando que el gradiente de la presión sea cero en los límites:

$$\frac{\partial p}{\partial n} = 0.$$

Y para las condiciones de límite del campo de velocidad u, Stam plantea que la velocidad en el límite es cero.

Stam adaptó el método de resolver las ecuaciones de Navier-Stokes con el objetivo de simular la dispersión de las densidades escalares arrastradas por el campo de velocidad del fluido. Esto es muy importante porque a menudo no se quiere visualizar el campo de velocidad directamente, pero si visualizar el efecto del campo de velocidad sobre una sustancia como tinta o humo. La siguiente ecuación da la solución del escalar campo de densidad *d:*

$$\frac{\partial d}{\partial t} = -\mathbf{u} \cdot \nabla d + k_d \nabla^2 - \alpha_d d + S_d.$$

Donde k_d es la constante de difusión, α_d es la tasa de disipación y S_d es el termino fuente. El término de advection[1], el de difusión y el de fuente son resueltos de igual manera que para el campo de velocidad, con S_d correspondiente al término de fuerzas

externas en las ecuaciones de Navier-Stokes. La diferencia es que en vez de la velocidad empujarse ella misma hacia delante, el paso de advection[1] ahora empuja el escalar campo de densidad hacia delante basándose en el campo de velocidad **u**. La disipación puede ser calculada de la siguiente forma:

$$(\mathbf{1} + \Delta t \alpha_d)\mathrm{d}'(x) = d(x).$$

Con arbitrarios pasos de tiempo Δt, el estable método de Stam intercambia exactitud por rendimiento, obteniendo resultados aceptables sin sacrificar la estabilidad. Este método captura los elementos visuales del flujo de los fluidos, mientras que mantiene la velocidad suficiente como para mantener interactivos frame-rates[3] en razonables tamaños de malla.

1.3 Métodos de simulación física

Recientemente, debido al avance en las investigaciones y el crecimiento del poder de procesamiento de las computadoras, los métodos han sido más enfocados en el comportamiento físico de los fenómenos que en los comportamientos definidos por el programador como se hacía en tiempos pasados (6). La ventaja de los métodos basados en la física de los fenómenos es que estos ofrecen más flexibilidad y menos ad hoc que los métodos tradicionales y se escala mejor para diferentes tipos de efectos.

Estos métodos son divididos generalmente en 3 partes principales, la simulación de la dinámica de los fluidos que describen los efectos; que generalmente se utilizan las ecuaciones de Navier-Stokes, descritas en la sección 1.2.2; para simular los aspectos de transferencia de masa y calor de los efectos que presentan combustión (efectos de pirotecnia), mediantes aproximaciones y variaciones del método descrito por Stam (1) y que se planteó en la sección 1.2.3,.La simulación de la combustión y la simulación de la turbulencia.

1.3.1 Simulación de los fluidos

Una forma de simular los fluidos del fuego es dado en (2), el flujo del combustible vaporizado y el flujo de los productos gaseosos calientes del fuego; son modelados independientemente, usando las ecuaciones de Navier-Stokes por separado y el

dominio computacional o área de la simulación es dividido en N^3 voxel[4] con un espacio uniforme *h*. Como resultado de esto se obtiene dos campos de velocidades separados, uno que controla el movimiento del combustible vaporizado y el otro, que controla los productos gaseosos. La densidad (tanto del humo como del hollín) y la temperatura son simuladas en campos separados controlados por el campo de velocidad resultante de los dos anteriores, o son definidos por el programador mediante curvas de comportamientos. Estas curvas especifican la cantidad de densidad o temperatura en varios tiempos después de ocurrido la reacción de combustión. Debido a la temperatura, el campo de velocidad de los productos gaseosos calientes es afectado de manera que los gases calientes tienden a elevarse debido a su flotabilidad. Esto es modelado en esta solución como fuerzas externas, proporcionales a la temperatura que actúan sobre el campo de velocidad.

En (7), se usan las ecuaciones para controlar el movimiento del sistema de tres gases, que consta de aire oxidante, gases combustibles y gases de escapes. La propagación del calor también es modelada. Como en el método usado en (2), el dominio de la simulación también es dividido en N^3 voxels[4] y resolver las ecuaciones resulta en el campo de velocidad que controla el movimiento del aire, que actúa como el fluido en la simulación. Los gases combustibles, los gases de escapes y el calor son simulados explícitamente en capos separados y advected[1] por el campo de velocidad del aire. Los tres campos son afectados por la disipación y el campo de temperatura es afectado por la difusión que simula la propagación del calor. Los gases combustibles, los gases de escapes y el calor afectan el campo de velocidad por turno, en términos de la fuerza de gravedad y una fuerza de flotabilidad. La fuerza de gravedad es proporcional a la cantidad de gases combustibles y gases de escapes, mientras que la fuerza de flotabilidad es proporcional a la temperatura. Esta última, es crucial para obtener la correcta forma de la llama.

En (8), las ecuaciones son resueltas implementando el método de fluidos estables en la GPU, modelando el flujo del fluido en el lugar donde el efecto tiene lugar. En ese caso se utilizó para simular fuego, pero esto puede ser extendido para cualquier otro efecto similar como la explosión. El campo de velocidad resultante de la simulación del fluido es usado para transportar el calor que es constantemente introducido y finalmente para darle movimiento a las partículas de fuego en la parte de visualización. A diferencia de los métodos anteriores, la simulación es afectada solamente por las fuerzas de flotabilidad y por los marcos de presiones pre calculado que son escaladas por la temperatura. La simulación del fluido es desarrollada en una maya 2D al contrario de (8), que es en un espacio 3D.

Otro método de simular los fluidos es como el que se usa en (9). En lugar de usar las ecuaciones de Navier-Stokes, se utiliza un modelo llamado LBM (Lattice Boltzmann Model o Modelo de Rejilla Boltzmann). Utilizar un LBM resulta en un campo velocidad 3D el cual es usado en la parte de visualización para mover las primitivas. El campo de velocidad es afectado por la dirección del viento y por objetos que no se queman y para modelar la generación de humo se utiliza un campo de temperatura.

En (6), con el objetivo de simular una explosión, se utiliza el método de fluidos estables para simular el movimiento del aire y de los gases calientes que están alrededor de la explosión. En este método, las partículas siguen el movimiento de combustibles particulares y productos de la combustión. Para la simulación del fluido se utiliza un modelo de fluido incomprensible como Navier-Stokes, pero en vez de usar un campo no divergente se ajusta este de manera que en algún momento o en algún lugar la masa no se conserve, es decir, el campo sea divergente:

$$\nabla \cdot \mathbf{u} \neq 0$$

Esto se hace con el objetivo de simular la expansión de los productos gaseosos. Inicialmente la temperatura en las regiones cercanas al centro de la explosión; se le dan valores extremadamente altos, después en los siguientes pasos de la simulación, la temperatura del calor adicionado va disminuyendo.

Otro usado y que no es muy basado en la física de los fluidos es el método de "Procedimiento de ruido (Noise)", el cual consiste en desarrollar un modelo de evolución de densidad de gas usando algún tipo de función aleatoria de ruido o procedimiento de textura. Una forma frecuente de hacer esto es utilizando el llamado "Perlinin noise" (…), el cual consiste en sumar funciones de diferentes amplitudes y frecuencias, cada una definida por una interpolación suavizada entre los valores aleatorios de una entrada de puntos. Esta función que genera los valores aleatorios tiene la característica de que para cada número dado debe retornar uno aleatorio que será siempre el mismo para el mismo dado.

Este método puede ser combinado con otras habilidades como por ejemplo en dependencia de las habilidades del animador, para crear turbulencias y capturar pequeñas escalas de detalles. Así como también para actualizar un sistema de partículas en una larga escala.

Este método tiene la desventaja de que requiere de una buena cantidad de implementación y características ad-hoc antes de producir simulaciones convincentes.

1.3.2 Simulación de la Combustión

Una forma de simular la combustión implícitamente en el fuego, es simulando el núcleo azul del fuego o de una llama, como plantea en (2), La superficie del núcleo azul aproxima la zona de reacción separando el gas combustible y los gases de escape y es modelado usando un método de entrada de nivel. También se usa un método de fluido fantasma (2) para simular la expansión que ocurre cuando el gas combustible es convertido a gases de escape en la reacción de combustión.

Otro método físico es dado en (8), se usa un modelo de combustión explícito para simular el proceso de combustión en cada celda de una maya que representa el dominio computacional. El parámetro de combustión es calculado basado en la cantidad de gas combustible y oxígeno existente en cada celda. El gas combustible, los gases de escapes y la temperatura son después actualizados basados en este parámetro, si la temperatura en la celda está por arriba de la temperatura con la cual el gas combustible se enciende. A raíz de esto, se puede simular diferentes reacciones de combustión tan solo variando los parámetros de ratio de encendido, que gobierna el por ciento de combustible que se quema en un segundo, y el parámetro de medidor de mezcla, que gobierna la cantidad de oxígeno requerido para quemar una unidad de gas de combustible.

El modelo de combustión usado en (4), es menos basado en la física y más dependiente de ad hoc[5], en (4) se modela la combustión indirectamente adicionando primitivas de visualizados en ensenadas y moviéndolas usando el campo vectorial obtenido del LBM. Cada primitiva contiene una cantidad de combustible y cuando el combustible es consumido, la primitiva de visualizado es eliminada de la simulación.

1.3.3 Simulación de la Turbulencia

Una desventaja del método de Stam para los fluidos estables es que, es esencialmente alta la cantidad de disipación numérica que presenta. Para corregir esto, en (1) se plantea combinar el método de fluidos estables con una fuerza de limitación rotacional (confinamiento de vórtices) la cual incrementa la turbulencia del campo de velocidad. El método de confinamiento de vórtice encuentra movimiento rotacional en cada celda de la maya del campo de velocidad chequeando las celdas adyacentes y calculando la fuerza de rotación la cual incrementa o mantiene el movimiento rotacional, esto reduciendo la cantidad de movimiento rotacional perdido

debido a la disipación numérica del método. De hecho, puede ser usado para crear aún más turbulencia en el campo de velocidad.

El confinamiento de vórtices ha sido utilizado en combinación con la simulación de fuego y explosiones con el fin de obtener y crear efectos más turbulentos y caóticos.

1.3.4 Extrusión Volumétrica

Desarrollar la implementación de un fluido en un domino 3D, es tan solo agregarle una componente más a la simulación hecha en un dominio plano 2D, pero desde el punto de vista computacional es más costoso el primero. Una implementación de la simulación del fluido en un dominio 3D puede alcanzar velocidades capases de ejecutarse en tiempo real para tamaños pequeños de la maya de simulación; pero para tamaños pequeños puede resultar poco práctica debido al costo computacional. En (1) se propone el uso de extrusiones volumétricas para derivar un campo de velocidad 3D de pocos campos de velocidades 2D, resultados de independientes simulaciones 2D; donde el campo de velocidad resultante es muy similar al obtenido si se simulara el fluido en un campo 3D.

Este método es más rápido que una completa simulación 3D, debido a que pocas simulaciones 2D requieren menos tiempo computacional. Los campos de velocidades 2D son visto como planos cruzados en un campo 3D (Fig. 2) y se define un método de interpolación que es el encargado de rellenar la región vacía que se encuentra entre los planos cruzados o campos 2D. Una posibilidad para esta interpolación es desarrollar un tilin[7] de la sección cruzada en una forma cilíndrica (Fig. 3), y realizar una interpolación cilíndrica a lo largo de un arco de radio constante del eje vertical entre 2 secciones cruzadas. Si el efecto tiene una simetría bastante aproximada, como por ejemplo las hogueras y las explosiones aéreas, esta interpolación sería bastante conveniente.

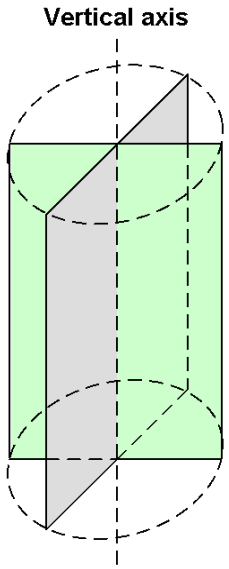

Figura 2 Secciones cruzadas 2D.

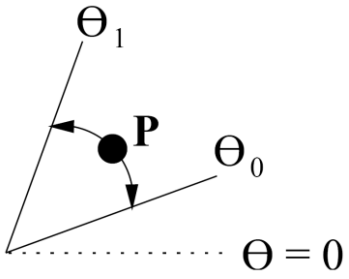

Figura 3 Interpolación cilíndrica (vista superior)

En (9) se usa este método para simular efectos de humo, explosiones y fuego. El dominio 3D es representado por un cilindro y una entrada de fluidos planos 2D que son calculados independientemente; que para obtener los valores del campo a través del dominio 3D, usa una interpolación cilíndrica, tomando valores de los planos más cercanos y combinándolos. Además de esto, se usa una distribución fractal estocástica o aleatoria con el objetivo de introducir características turbulentas de pequeñas escalas. Este método es más rápido pero limita el número de fenómenos que se pueden simular y es menos exacto.

1.3.5 Autómata celular

Un **Autómata Celular** (A.C) es un modelo matemático que modela a un sistema dinámico que evoluciona en pasos discretos. Es adecuado para modelar sistemas naturales que puedan ser descritos como una colección masiva de objetos simples que interactúen localmente unos con otros. Son sistemas descubiertos dentro del campo de la física computacional por John Von Neumann en la década de los 50s.

Un A.C. consiste de:

Una *rejilla* o *cuadriculado* (*lattice*) de enteros (conjunto $\mathbb{Z}$) infinitamente extendida, y con dimensión $d \in \mathbb{Z}^+$. Cada celda de la cuadrícula se conoce como célula.

1. Cada célula puede tomar un valor en $\mathbb{Z}$ a partir de un *conjunto finito de estados k*.
2. Cada célula se caracteriza por su *vecindad*, un conjunto finito de células en las cercanías de la misma.
3. De acuerdo con esto, se aplica a todas las células de la cuadrícula, una *función de transición* (f) que toma como argumentos los valores de la célula en cuestión y los valores de sus vecinos, y regresa el nuevo valor que la célula tendrá en la siguiente etapa de tiempo. Esta función *f* se aplica, como ya se dijo, de forma homogénea a todas las células, por cada paso discreto de tiempo (11).

Dentro del ámbito de los A.C., se pueden implementar numerosas condiciones de frontera, de acuerdo a lo que el problema real requiera para su modelado. Por ejemplo:

1. **Frontera abierta**. Se considera que fuera de la lattice residen células, todas con un valor fijo. En el caso particular del juego de la vida y de otros A.C. con dos estados en su conjunto *k*, una frontera se dice *fría* si las células fuera de la frontera se consideran muertas, y *calientes* si se consideran vivas.
2. **Frontera periódica**. Se considera a la lattice como si sus extremos se tocaran. En una lattice de dimensión uno, esto puede visualizarse en dos dimensiones como una circunferencia. En dimensión dos, la lattice podría visualizarse en tres dimensiones como un toroide.
3. **Frontera reflectora**. Se considera que las células fuera de la lattice *reflejan* los valores de aquellas dentro de la lattice. Así, una célula que estuviera junto al borde de la lattice (fuera de ella) tomaría como valor el de la célula que este junto al borde de la lattice, dentro de ella.
4. **Sin frontera**. Haciendo uso de implementaciones que hagan crecer dinámicamente el uso de memoria de la lattice implementada, se puede asumir que cada vez que las células deben interactuar con células fuera de la lattice, esta se hace más grande para dar cabida a estas interacciones. Obviamente, existe un límite (impuesto por la memoria disponible) para esta condición. Es muy importante no confundir esta condición de frontera con la definición original de A.C. cuya lattice es inicialmente infinita. En el caso de un A.C. sin frontera, la lattice comienza con un tamaño definido y finito, y conforme se requiera va creciendo en el tiempo, lo cual no lo hace necesariamente un

modelo más cercano a la realidad, pues si se inicializara la lattice aleatoriamente, con esta condición sólo se pueden inicializar las células dentro de la lattice inicial finita, mientras que en el caso de la definición original, en teoría todas las células de la lattice infinita deberían ser inicializadas.

El modelar un sistema del mundo real por medio de un *Autómata Celular*, requiere que se conozca al menos su comportamiento global. Si conocido este comportamiento se quiere deducir un conjunto de reglas de evolución local que lo genere, entonces se desea desarrollar el autómata por el *Problema Inverso*.

De lo contrario, si se desea primero experimentar y ajustar una *Regla de Evolución* pseudo-aleatoria hasta lograr un comportamiento similar al del sistema real, entonces se desea desarrollar el autómata por el *Problema Directo*. No obstante, se puede lograr algo intermedio, a partir de comportamientos locales del sistema real, construir una regla de evolución local y ponerla a prueba para determinar si se logra un autómata que modele el comportamiento del sistema global, a esto se el denomina el *Problema Intermedio*.

Dependiendo de la naturaleza compleja de un sistema y de la posibilidad de identificar estados locales y reglas generales de evolución, se podrían simular comportamientos por medio de *Autómatas Celulares*; por ejemplo, los mundos y sistemas enunciados a continuación son susceptibles a un modelamiento por esta técnica: Simulación de tráfico automotor, virus, glóbulos, epidemias, bacterias, contaminación, ecosistemas, evolución galáctica, flujo de electrones, acción y reacción, medios granulares y gases de Fermi entre otros(12).

1.4 Métodos de simulación visual

El resultado de las simulaciones física de los efectos pueden ser visualizados de diferentes maneras pero la mayoría tiene características semejantes, debido a esto se han agrupado en dos métodos generales de los cuales se derivan muchos otros. Estos métodos son los Sistemas de Partículas y el de Visualizado Volumétrico (Volume Rendering).

1.4.1 Sistema de Partículas

Los sistemas de partículas han sido usados tanto para describir técnicas de visualizado como para describir tipos de animaciones específicas. Debido a esto, su definición depende del tipo de aplicación que se va a usar. El criterio general de definición para cualquier tipo de aplicación es el siguiente (10):

1. Es una colección de partículas: El sistema está compuesto por una o más partículas individuales, donde cada una de ellas tienen un conjunto de atributos que de manera directa o indirectamente afectan el comportamiento de la partícula. A menudo las partículas son primitivas gráficas como puntos o líneas, pero en realidad no existe límite para esto.
2. Define atributos o propiedades estocásticas: Otra característica en común que tienen los tipos de sistemas de partículas es que introducen en ellas algún tipo de elementos o variables aleatorias, los cuales pueden ser usados para controlar la variación aleatoria de la velocidad, o el cambio de color, incluso su posición. Usualmente estos elementos aleatorios son controlados por algún tipo de límite estocástico predefinido, como límites o tipos de distribución.
3. Tienen un ciclo de vida: Cada partícula tiene tres estados distintos en su tiempo de vida, Nacimiento, en el cual cada partícula en el sistema es generada de manera aleatoria, así como también puede ser la generación de su forma. Vida, en el cual los atributos de las partículas cambian al pasar del tiempo. estos atributos pueden depender del tiempo o de otro atributo o de ambos. Y Muerte, que es cuando el tiempo de vida de la partícula llego a su fin o cuando está fuera de los límites o cuando por ejemplo el color de la partícula se torna negro lo que hace que no sea visible.
4. Forma de las partículas: Las partículas pueden tener cualquier tipo de forma, forma esférica, forma de caja, punto, pero generalmente tienen forma de un plano como se muestra en la (Fig. 4).

Este método puede resultar tener un amplio código fuente, por eso es importante diseñar bien la estructura de datos que representará nuestra partícula para mejorar el rendimiento del sistema. En el procesamiento de miles de partículas, un milisegundo de más, puede agrandar el tiempo de ejecución, por eso, cuando se diseña un sistema de partículas lo primero que se debe tener en cuenta es que los sistemas de partículas incrementan grandemente el número de polígonos visibles.

La mayoría de los sistemas de partículas usan cuatro vértices y dos triángulos para representar una partícula (Fig. 4). Esto conlleva a que un sistema que use más de

2000 partículas incremente la cantidad de polígonos en la escena a más de 4000 que hay que visualizar y re calcular los datos dentro del buffer en cada frame.

Una solución para optimizar un sistema de partículas, es utilizar lo menos posible operaciones de memoria, como asignación y eliminación de memoria a objetos. Cuando una partícula muere, no eliminarla de memoria sino solamente marcarla como "muerta", así cuando todas las partículas del sistema mueran entonces se liberan todas juntas (10). Esto permite reutilizar partículas muertas, es decir memorias asignadas, como por ejemplo en la simulación de flujos constantes como el fuego, puedes revivir partículas ya muertas tan solo inicializando sus parámetros.

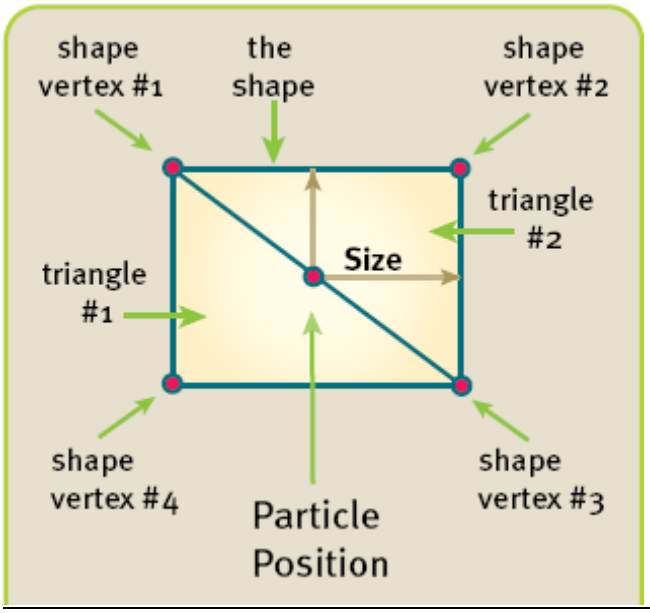

Figura 4 Forma de la partícula.

Otro aprovechamiento que permite aumentar el rendimiento de los sistemas de partículas es el uso de los LODs (Levels Of Detail). Este método está basado en una subdivisión física que genera una jerarquía aproximada de modelos en movimientos o niveles de detalles de la simulación (SLODs). En cada paso de tiempo se actualiza los SLODs y es seleccionando el más apropiado SLOD para reducir el costo computacional. La idea general es que se sustituye un grupo de partículas por una que represente toda el área abarcada por el grupo en dependencia de cuán lejos sea la simulación de acuerdo con la cámara. Mientras más lejos esté, menos nivel de detalle se necesita, por tanto menos partículas. Mientras más cerca este, mayor nivel de detalle, y por tanto mayor número de partículas.

En (9) es usado un sistema de partículas implementado en la GPU (Graphic Processor Unit) para visualizar fuego. Por cada partícula, es reconstruido un vector de velocidad

en la posición actual usando un campo de velocidad 2D de la simulación del fluido y el método de extrusión volumétrica, que es utilizado para convertir la simulación de 2D a 3D. Esta velocidad es después usada para mover la partícula según la dirección del flujo y la nueva posición de la partícula es almacenada. Finalmente la partícula es renderizada[6] como un punto donde el color es modificado de acuerdo con una especifica tabla de colores o usando los flujos de densidad o temperatura.

1.4.2 Representación Volumétrica (Volumen Rendering)

Volumen Rendering es un método para visualizar imágenes de volúmenes (datos 3D) de manera completa sin el uso de segmentación. Este método tiene la ventaja de que toma en cuenta la información de los datos en su totalidad, y de esta manera logra una representación 3D muy completa.

La información que generalmente se requiere son imágenes de cortes transversales de algún volumen. Cada corte transversal está formado por un arreglo bidimensional de valores de intensidad. El volumen se forma agrupando las imágenes, es decir considerando que cada punto de la imagen tiene coordenadas (x,y), si se agrega la coordenada z, por medio de el número de corte se obtiene un conjunto de coordenadas tridimensionales confinadas en un paralelepípedo. Cada coordenada se representa por un voxel[4], el cual es un cubo cuyo ancho, alto y espesor, tiene tamaño de un píxel. El valor del voxel[4] que se encuentra en el punto (x,y,z) tendrá el valor del punto (x,y) en el corte z.

Para la visualización de efectos especiales, se han usado variaciones de este método. Una de estas es dada en (8), donde se remplaza cada voxel[4] por un polígono semitransparente donde el nivel de trasparencia es controlado por la densidad del fluido en cada voxel[4]. La cantidad de gas combustible es mostrado en amarillo y la zona de reacción donde ocurre la combustión es mostrada en rojo. La intensidad del color rojo está dada por la cantidad de combustión que ocurre en cada voxel[4].

Un método similar se plantea en (1). En vez de remplazar cada voxel por un polígono semitransparente, se posiciona una textura en el centro de cada voxel y se visualiza la textura usando la GPU. El color de la textura es calculado usando la distribución de la temperatura resultado de la simulación mediante la fórmula de Planck para las radiaciones de cuerpo-negro que se describió anteriormente en la sección 1.1.1.

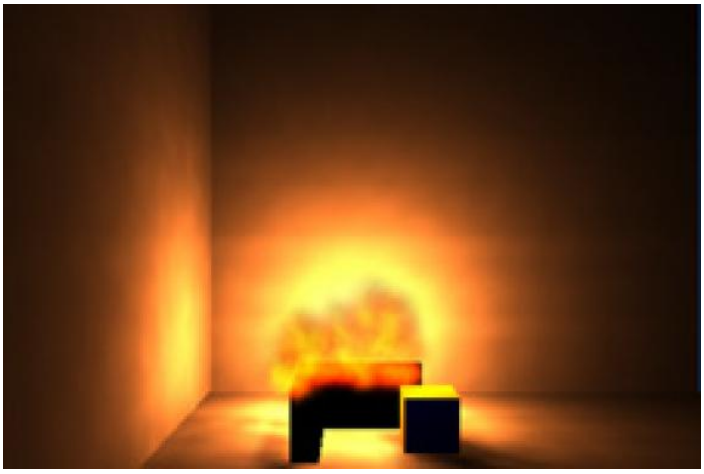

Figura 5 Simulación de fuego usando *"Volume Rendering"*.

Tradicionalmente este método ha sido muy lento para su trabajo en aplicaciones de tiempo real, pero ha sido capaz de alcanzar altas calidades de visualización. Recientemente su implementación para su uso en tiempo real ha sido gracias a la utilización de la GPU, pero aun así es costoso.

1.4.3 Metaball

Metaball es el nombre de una técnica de gráficos realizada por ordenador para simular interacción orgánica entre diferentes objetos n-dimensionales (como gotas de mercurio mezclándose por su superficie) y fue inventado por Jim Blinn a principios de los años 1980.

Cada metaball es definida en función de n-dimensiones (es decir para tres dimensiones, f (x, y, z); los metaballs tridimensionales tienden a ser los más comunes). Un valor de umbral también es elegido, para definir un volumen sólido.

Entonces, $\sum_{i=0}^{n} metaball\ i\ (x, y, z) \leq Umbral$ representa el volumen encerrado por una superficie entre dos o más metaballs en (x, y, z).

Una función típica elegida para metaballs es:

***F*(x, y, z) = 1 / ((x − x0) 2 (y − y0) 2 (z − z0) 2)**; donde **(x0, y0, z0)** es el centro del metaball. Sin embargo, debido a la división, es computacionalmente muy exigente. Por esta razón son comúnmente usadas aproximaciones a funciones de polinomios (13).

Un ejemplo de la utilización de los Metaball es en la simulación de las nubes en un entorno virtual como se observa en la figura6.

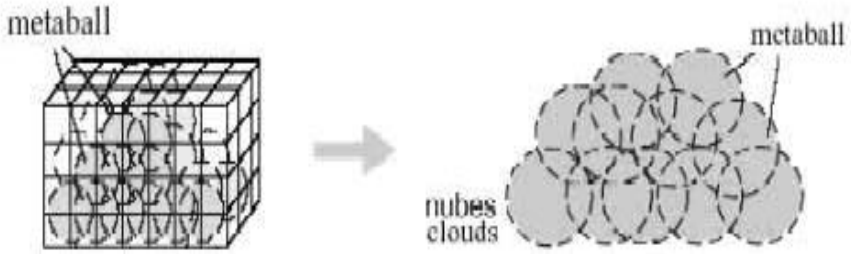

Figura 6 Rendering de las nubes utilizando *"Metaball"*

1.5 Conclusiones

Mundialmente el uso y desarrollo de los diferentes métodos tanto para la simulación física como para la visualización, esta cada día en aumento. De esto se debe las apariciones de nuevos métodos, nuevas aproximaciones y nuevas técnicas de modelado para objetos sin forma como son los fluidos.

Capítulo 2: Soluciones Técnicas

Introducción

En el presente capítulo se propone soluciones técnicas para crear efectos de pirotecnia como Explosiones y Fuego partiendo de la dinámica de los fluidos en la parte física y del sistema de partículas para la visualización.

2.1 Método de Visualización

Como solución se propone el uso de un sistema de partículas genérico como método de visualización para los diferentes efectos especiales. Un sistema de partículas capaz de asimilar cualquier modelo matemático que responda al comportamiento de cada partícula y que defina un efecto en sí. Siendo este sistema lo suficientemente extensible como para poder usar un solo tipo de definición de sistema de partículas para una serie de efectos especiales.

Se ha escogido este método de visualización porque en comparación con los planteados en el capítulo anterior, tiene mejores características para el trabajo en tiempo real, que es uno de los objetivos principales de la investigación, además de su fácil uso y manejo.

El sistema estará compuesto por:

1. Una estructura partícula: que describirá aquellos atributos necesarios en cada partícula. Además será capaz de soportar tantas definiciones de estructuras de partículas como el desarrollador quiera, dando vía a la extensibilidad del sistema.
2. Métodos de inicialización: que se encargarán de inicializar los atributos de las partículas según el efecto escogido.
3. Modelos matemáticos: que empaquetarán la física que rige el movimiento de cada partícula para cada efecto especial.
4. Será capaz de hacer un control centralizado del conjunto de partículas independientemente del modelo que se haya definido.

La principal característica de esta solución y que contribuye a la realización de nuestra solución, es que se desarrollará usando la estrategia de **template,** que es propia del lenguaje que se utilizará (C++). Debido a que los template añaden la posibilidad de definir que comportamiento (política de acción) debe exhibir cada partícula en tiempo de pre compilación. Así como también definir el tipo de partícula ha visualizar, a parte de los grandes beneficios que brinda para la optimización del sistema.

En el sistema trabajarán en conjunto la CPU con GPU, desarrollando la simulación física en la GPU y las operaciones de visualización en la CPU.

Se ha escogido el uso de los aceleradores gráficos (GPU) como una de las vías de implementación para la simulación física debido a que contribuye enormemente al aumento de la velocidad de la simulación. Implementar el método estable de Stam a través de shaders (especialmente con fragments programs), proporcionaría un alto nivel de programación en paralelo, además se conoce que la GPU tiene más capacidad de procesamiento que la CPU, ya que solamente está destinada para el gráfico y puede procesar cientos de gigaflops para el cálculo de un simple punto flotante mientras que en la CPU solamente serían 12 gigaflops.

2.2 Simulación Física

Para la simulación física de los fluidos se trabajará con el método de fluidos estables desarrollado por Jos Stam y que fue explicado en el capítulo anterior. Este método se ha escogido debido a sus características de fácil manejo y comprensibilidad siendo una solución para tiempo real.

Para la simulación de fuego se empleará el modelo descrito en (1), donde se modela la evolución del campo de velocidad, que representa el flujo de aire, usando el método estable de Stam y los campos de densidades escalares, que son el campo de gas combustible, el campo de gas de escape y el campo de temperatura, a través del campo de velocidad.

Campo de velocidad

El campo de velocidad será calculado mediante la siguiente modificación de la ecuación de Navier-Stokes:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{F}$$

$$\nabla \cdot \mathbf{u} = \mathbf{0}$$

Donde el término de viscosidad ha sido eliminado debido a que la viscosidad del aire es despreciable. El término de la fuerza **F** será igual a:

$$\mathbf{F} = f_{\mathbf{vortice}} + f_{\mathbf{gravedad}} + f_{\mathbf{flotabilidad}}.$$

Donde el primer término de la parte derecha es el valor de la fuerza que crea la turbulencia y será igual a:

$$f_{\mathbf{vortice}} = \epsilon h(\mathrm{N} \times \omega)$$

$$\mathrm{N} = \frac{\nabla|\omega|}{|\nabla|\omega||}.$$

$$\omega = \nabla \times \mathbf{u}.$$

Donde ϵ controla la potencia de la fuerza que causa turbulencia y *h* es la distancia entre dos celdas. El segundo término es el valor de la fuerza de gravedad:

$$f_{gravedad} = f_g\ (g + a)\begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}.$$

Donde f_g controla la potencia de la fuerza de gravedad, mientras que *g* y *a* son respectivamente la cantidad de gas combustible y la cantidad de gas de escape en cada celda.

El tercero y último término de la ecuación es la fuerza de flotabilidad, que causa que los aires calientes se eleven:

$$f_{flotabilidad} = f_f(T - T_{ambiente})\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Donde f_f es la constante de flotabilidad, *T* la temperatura en la celda y $T_{ambiente}$ es la temperatura ambiente.

Campos de densidades

Para describir la evolución de los campos de densidades se utilizará las ecuaciones descritas en (1):

$$\frac{\partial g}{\partial t} = -(\mathbf{u} \cdot \nabla)g + k_g \nabla^2 g - \alpha_g g + S_g + C_g.$$

$$\frac{\partial a}{\partial t} = -(\mathbf{u} \cdot \nabla)a + k_a \nabla^2 a - \alpha_a a + C_g.$$

$$\frac{\partial T}{\partial t} = -(\mathbf{u} \cdot \nabla)T + k_T \nabla^2 T - \alpha_T T + C_T.$$

Donde el primer termino de cada una de ella representa el movimiento causado por el campo de velocidad al campo que se aplica. El segundo es el término de difusión, que simula la tendencia de la densidad a esparcirse. El tercer término gobierna la disipación del campo de densidad.

El término S_g de la primera ecuación es la fuente de gas, y los términos *Cg, Ca* y C_T son los relativos a la combustión:

$$C_g = -\frac{C}{b}.$$

$$C_a = C(1 + \frac{1}{b}).$$

$$C_T = T_0 C.$$

Donde *C* que es el parámetro de combustión, cumple con la siguiente condición:

$$C = \begin{cases} rbg & \textbf{if } T > T_{humbral} \\ 0 & \textbf{if } T \leq T_{humbral} \end{cases}.$$

El término *r* es el ratio de quemado, representa que tan rápido se quema el combustible, *b* representa la cantidad de oxígeno que se necesita para quemar una unidad de combustible y *g* es la cantidad de gas combustible presente en la celda.

2.3 Herramientas de desarrollo y lenguaje utilizado

Las herramientas que se proponen en este epígrafe serán puestas en marcha en todas las fases de desarrollo del módulo, tal como el lenguaje que se referencia.

2.3.1 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 ha 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 realizan en C++, además de ser un lenguaje con posibilidad de desarrollo en plataforma libre.

2.3.2 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), la participación de Rational en 1998 con respecto a los ingresos del mercado es mayor que las participaciones de los siguientes cuatro competidores directos combinados. 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.

2.3.3 C++

Existen principalmente tres lenguajes que se utilizan para desarrollar aplicaciones gráficas profesionales en 3D: Lenguaje Ensamblador, C y C++, por ser los que con más velocidad ejecutan el código (menor costo de ejecución del programa). A éstos se ha unido recientemente el Java como una opción para el desarrollo de este tipo de aplicaciones.

Es decisión de la entidad cliente Simpro, implementar este proyecto mediante el lenguaje C++, que ha estado en su línea de trabajo con magníficos resultados.

Es el lenguaje en que se tiene mayor experiencia por parte del equipo de SIMPRO-UCI, futuros desarrolladores del sistema que ocupa este proyecto.

Si se estudian las características de este lenguaje, se puede apreciar lo acertado de la elección, dado que C++ es un lenguaje de programación de propósito general, especialmente indicado para la programación de sistemas por su flexibilidad y su potencia.

2.3.4 Cg de NVidia

Cg es un lenguaje de alto nivel que hace posible la creación de diferentes efectos especiales. Diseñado para que el programador pueda controlar una serie de atributos gráficos usando un hardware gráfico programable, controlando estos atributos con una rapidez increíble. Este leguaje provee al programador con una completa plataforma de programación que es fácil de usar y que soporta diferentes plataformas; para sistemas operativos como Windows, Linux, Macintosh OS X y plataformas de consolas como Xbox; y es compatible con OpenGL y DirectX,. A diferencia de otros lenguajes como MOD o HLSL que fueros diseñados para soportar OpenGL y DirectX respectivamente.

Debido a sus características todos los shaders que se utilizarán en este Módulo serán implementados haciendo uso de este leguaje de programación.

Conclusiones

En este capítulo se abordo las soluciones técnicas que se irán desarrollando en el transcurso del los siguientes capítulos para darle solución a nuestro problema. Como parte de la simulación física se retomará los métodos de Stam y en la parte de simulación física se realizará mediante el sistema de partículas los cuales tienen un alto grado de visualización en tiempo real.

Capítulo 3: Características del Sistema

Introducción

En este capítulo se obtiene una visión más específica del sistema que se va a desarrollar, partiendo fundamentalmente de las soluciones técnicas. De inicio se obtiene una panorámica desde el punto de vista del negocio dando continuidad a crear la concepción práctica del producto a elaborar partiendo de las necesidades del cliente desde el punto de vista de que es lo que debe hacer el sistema.

3.1 Reglas del negocio

El fichero de textura a cargar para las partículas debe ser de extensión BMP, en caso de no existir el fichero en el camino indicado, ser de otro formato, estar corrupto o no tener el modelo textura no constituye un error, solo se debe indicar lo sucedido y cargar el modelo sin textura.

El programador que emplee esta herramienta debe tener conceptos previos de simulación de partículas y fundamentalmente de la dinámica de los fluidos.

3.2 Modelo de Dominio

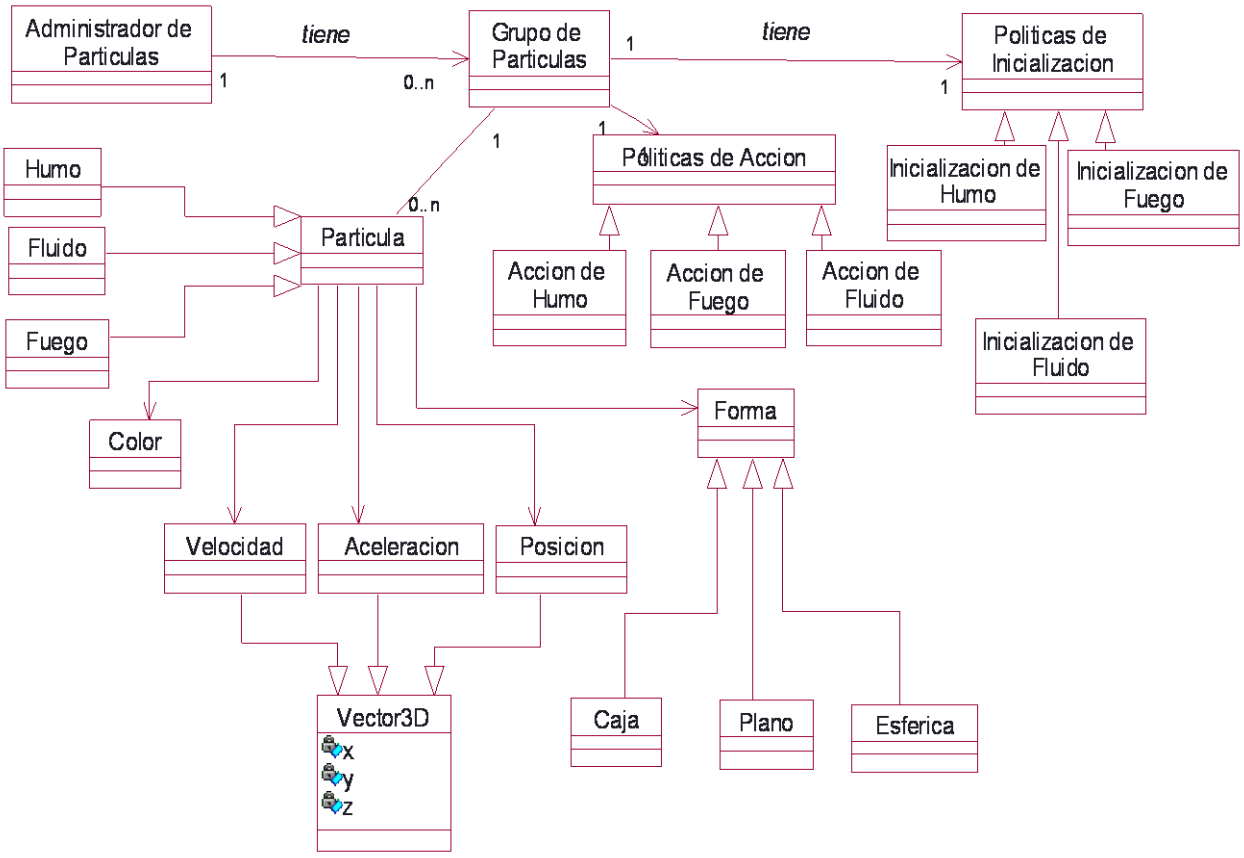

Figura 7 Diagrama del Modelo del Dominio.

3.3 Glosario de Términos del Dominio

Partícula: es un registro que contiene algunos atributos.

Aceleración: es un vector de aceleración velocidad que utiliza el movimiento de la partícula para poder simular los efectos.

Posición: es un vector que proporcionará la posición de la partícula en el espacio.

Velocidad: es un vector velocidad que se utiliza en el movimiento de la partícula.

Forma: Puede ser de cualquier tipo, esférica, forma de caja, forma de un plano, etc.

Color: Define el color como un RGB (para los píxeles). Puede usarse para almacenar el índice de un arreglo de bitmaps[7] creado previamente para colorear cada partícula.

Vector: Vector de 3 componentes flotantes (x, y, z).

3.4 Captura de requisitos

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

3.4.1 Requisitos funcionales

1- Crear estructura de trabajo vértice.
2- Crear estructura de trabajo color.
3- Definir atributos de partículas.
4- Crear partícula.
5- Definir la forma de la partícula.
6- Seleccionar fichero de textura.
7- Verificar extensiones de fichero de textura.
8- Cargar textura.
9- Definir efecto a utilizar.
10- Definir en qué parte se hará la simulación [CPU, GPU].
11- Definir cantidad de partículas a utilizar en el efecto.

12- Calcular elementos necesarios que se utilizarán en la simulación de las partículas (pre acción).

13- Actualizar las partículas.

14- Crear las estructuras de la simulación física.

15- Inicializar las variables con los valores físicos (difusión, combustión, presión, etc.).

16- Definir los shaders que se utilizarán.

17- Verificar las extensiones

18- Cargar los shaders.

19- Dibujar las partículas (render).

20- Modificar las partículas con nuevos atributos.

21- Ejecutar un efecto para su visualización.

22- Eliminar las estructuras realizadas en la simulación física.

23- Eliminar las estructuras realizadas en la simulación del visual.

24- Localizar un efecto.

3.4.2 Requisitos no funcionales

- Usabilidad**:** Los futuros usuarios del sistema serán programadores con conocimientos básicos con respecto al tema de los efectos de especiales de pirotecnia.
- Rendimiento: Debe ser una aplicación en tiempo real donde tenga gran alto de velocidad de procesamiento de los cálculos, tiempo de respuesta y recuperación del sistema.
- Soporte: Debe ser compatible con la plataforma Windows.
- Software: Sistema operativo Windows.
- Hardware**:** Compatibilidad con tarjetas gráficas de la familia NVIDIA, y tarjeta de video de más de 128 megabyte de RAM.
- Diseño e implementación**:** Debe trabajar con la biblioteca gráfica de OpenGL y además de emplear los Estándares de codificación del proyecto Simpro. Se harán llamadas a dichas bibliotecas desde Leguaje C++. Se regirá por la filosofía de Programación Orientada a Objetos.

3.5 Modelo de Casos de Uso

En este epígrafe se darán a conocer los actores del sistema a desarrollar, y a partir de los requisitos funcionales del epígrafe anterior se obtendrán los casos de usos del sistema con sus respectivas descripciones.

3.5.1 Definición de los actores del sistema

Tabla 1 "Justificación de los actores"

Actores	Justificación
Programador	Es el que se beneficiará con las funcionalidades que ofrece el módulo "Efectos de pirotecnia" donde obtendrá: crear efecto, localizar efecto, etc.

3.5.2 Casos de usos del sistema

1. Crear efecto.
2. Cargar Shader.
3. Ejecutar efecto.
4. Localizar efecto.
5. Actualizar efecto
6. Modificar efecto.
7. Eliminar efecto.

3.5.3 Diagrama de casos de usos

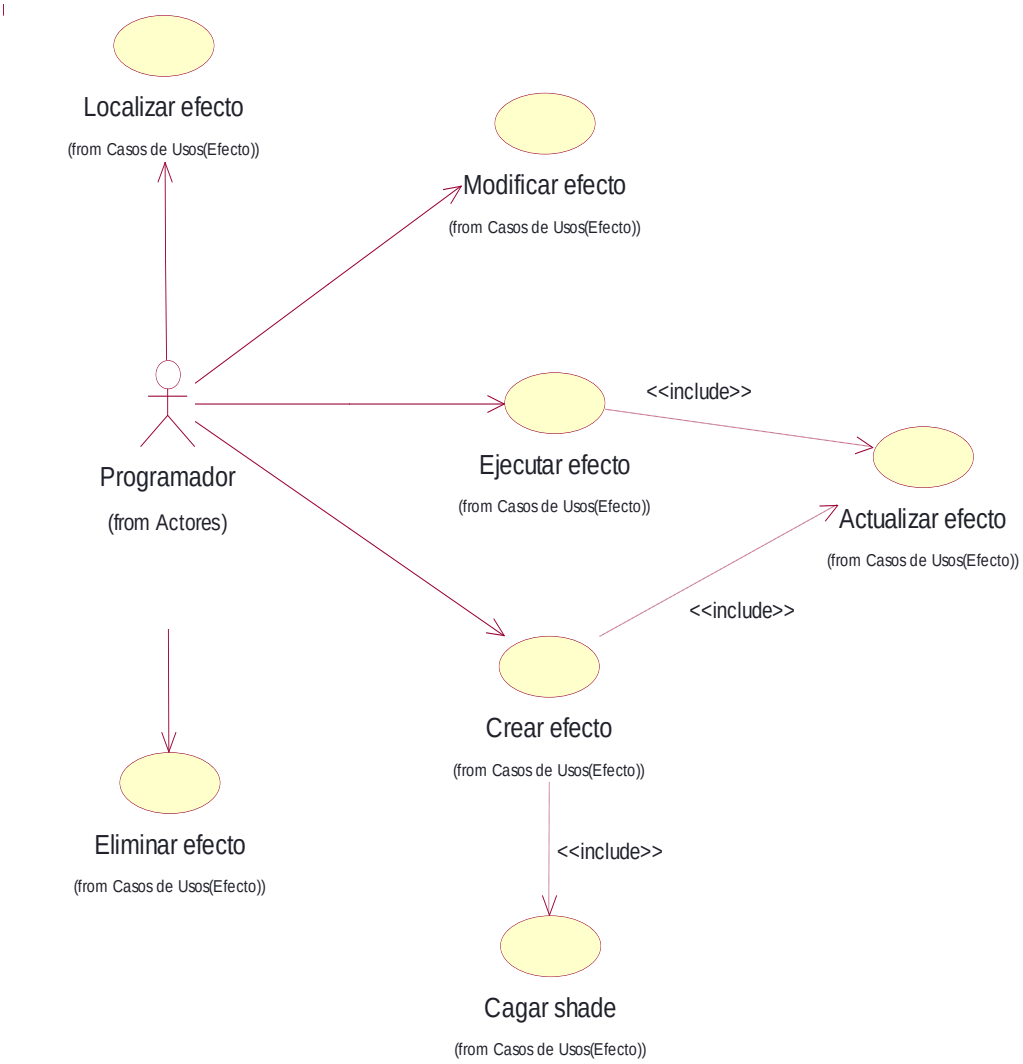

Figura 8 Diagrama de Casos de Usos.

3.5.4 Descripción de los casos de uso en formato expandido

Tabla 2 CU "Crear efecto"

Nombre del caso de uso	**Crear efecto.**
Actores	Programador
Propósito	Crear los efectos que pueden ser fuego, humo o explosiones.
Resumen: Se inicia cuando el actor solicita crear un efecto pasándole la cantidad de partículas que se quieren emitir, el tipo de partícula, los valores de los coeficientes (Temperatura, Gas, Combustible), así como la textura que se asignará, También pedir si se realizará en la "CPU" o "GPU". De ocurrir algún error en el proceso se le informa al Programador. El CU finaliza cuando el efecto elegido, es creado con todos sus objetos y variables.	
Referencias	R5, R6, R7, R8, R9, R10, CU2 *(include),* CU5 *(include).*
Precondición	-
Poscondición	Efecto creado.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita crear efecto pasando el nombre del efecto, la cantidad de partículas que se quieren emitir y los valores físicos y si es en la "CPU" o "GPU".	1.1- Crea la lista de partículas con la cantidad de partículas.
	1.2- Inicializa el origen del sistema de partículas.
	1.3- Verifica extensión de textura y existencia del fichero de textura.
	1.4- Se carga la textura de la partícula.
	1.5- Crea la simulación física con los parámetros físicos.

	1.6- Si es en la CPU Carga los Shader.
Curso alterno:	
Linea 1.3	
	1.3- Si la extensión no es la correcta muestra un mensaje de error.
Prioridad: Crítico.	

Tabla 3 CU"Cargar shader"

Nombre del caso de uso	**Cargar shader.**	
Actores		
Propósito	Cargar los shader.	
Resumen: El Caso de Uso se inicia cuando el CU "Crear efecto". Solicita cargar los shader de un determinado tipo de efecto.		
Referencias	R14, R15, R16	
Poscondición	Shader cargado.	
Curso normal de los eventos:		
Acción del actor		Respuesta del sistema
1- Se solicita cargar shader.		1.1- Verifica la existencia del fichero.
		1.2- Verifica si la extensión es la correcta.
		1.3- Se carga el shader.
Curso alterno:		
Línea 1.1		
		1.1- Si el fichero no existe muestra un mensaje de error.

Línea 1.2	
	1.2- Si la extensión no es la correcta muestra un mensaje de error.
Prioridad: Secundario.	

Tabla 4 CU "Localizar efecto"

Nombre del caso de uso	Localizar efecto.
Actores	Programador.
Propósito	Localizar un efecto determinado.
Resumen: El Caso de Uso se inicia cuando el actor, solicita localizar un efecto que esté en el entorno virtual.	
Referencias	R22.
Poscondición	Efecto localizado.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema.
1- Solicita localizar efecto.	1.1- Una vez que se tenga la posición del efecto, se toman las coordenadas del efecto y se retornan.
Curso alterno:	
Prioridad: Secundario.	

Tabla 5 CU" Modificar efecto"

Nombre del caso de uso	Modificar efecto**.**
Actores	Programador.
Propósito	Modificar un efecto en cualquier momento.
Resumen: El Caso de Uso se inicia cuando el actor solicita modificar un efecto pasándole los valores que serán cambiados.	
Referencias	CU6*(include)*
Precondición	Que el efecto este creado.
Poscondición	Efecto modificado.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Se solicita modificar efecto donde se pasan los valores que serán modificados puede ser la cantidad de partículas, posición y los valores físicos.	1.1- Se cambian los valores del efecto por los nuevos.
	1.2- Se actualizan todos los nuevos valores.
Curso alterno:	
Prioridad: Secundario.	

Tabla 6 CU "Actualizar efecto"

Nombre del caso de uso	Actualizar efecto**.**
Actores	
Propósito	Actualizar todas las variables de las partículas como son la posición, velocidad etc.
Resumen: El Caso de Uso se inicia cuando el CU "Crear efecto" o el CU "Ejecutar efecto" solicitan actualizar todas las partículas que se emitirán en un efecto, disminuyendo los valores del tiempo de vida.	
Referencias	R12
Precondición	-
Poscondición	Todas las partículas estén actualizadas.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1 – Indica actualizar los valores.	1.1- Se le da una acción previa a las partículas.
	1.2- Se eliminan de la simulación todas las partículas que estén en estado muerta.
	1.3- Se disminuye el tiempo de vida de las partículas.
Prioridad: Crítico.	

Tabla 7 CU "Ejecutar efecto"

Nombre del caso de uso	Ejecutar efecto**.**
Actores	Programador.
Propósito	Ejecutar un efecto.
Resumen: El CU se inicia cuando el actor solicita ejecutar el efecto, donde el efecto será visualizado.	
Referencias	CU6*(include)*
Precondición	Que el efecto este creado.
Poscondición	Efecto ejecutado.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Se solicita ejecutar el efecto.	1.1- Si el efecto esta en true se visualiza.
	1.2- Se actualizan los valores físicos.
	1.3- Se actualizan los valores del sistema de partículas.
	1.4- Se dibujan las partículas que se encuentren en el sistema de partículas.
Curso alterno:	
Linea 4	
	4- Si el efecto esta en false se pone en true y se visualiza.
Prioridad: Crítico.	

Tabla 8 CU "Eliminar efecto"

Nombre del caso de uso	Eliminar efecto.
Actores	Programador
Propósito	Eliminar el efecto que se encuentra en proceso.
Resumen: El Caso de Uso se inicia cuando el actor, solicita eliminar el efecto. Donce el sistema destruye todos los valores del sistema de partículas y de la simulación física.	
Referencias	R21, R22
Precondición	Que el efecto este creado.
Poscondición	Efecto eliminado.
Curso normal de los eventos:	
Acción del actor	Respuesta del sistema
1- Solicita eliminar efecto.	1.1- Destruir los valores que estén la simulación visual.
	1.2- Destruir los valores que estén la simulación física.
Prioridad: Secundario.	

Conclusiones

En este capítulo se definieron las funcionalidades del sistema, se definieron los casos de usos que darán vida al sistema como también se describieron los CU en formato expandido para tener una visión más clara de ellos.

Capítulo 4: Diseño del Sistema

Introducción

A continuación se presentarán los diagramas de clases de diseño el cual tendrá dos Paquetes uno de "Simulación Visual" y el otro "Simulación Física" con las relaciones existente entre ellas que pueden ser de agregación, generalización/especialización y composición. Además se presentan los diagramas de secuencia de los casos de usos que intervendrán en el primer ciclo de desarrollo de software.

Modelo de Diseño

Los artefactos de diagrama de clases de diseño y los diagramas de secuencias por casos de usos del sistema, se darán a conocer en los siguiente sub epígrafes.

4.1 Diagramas de Clases de Diseño

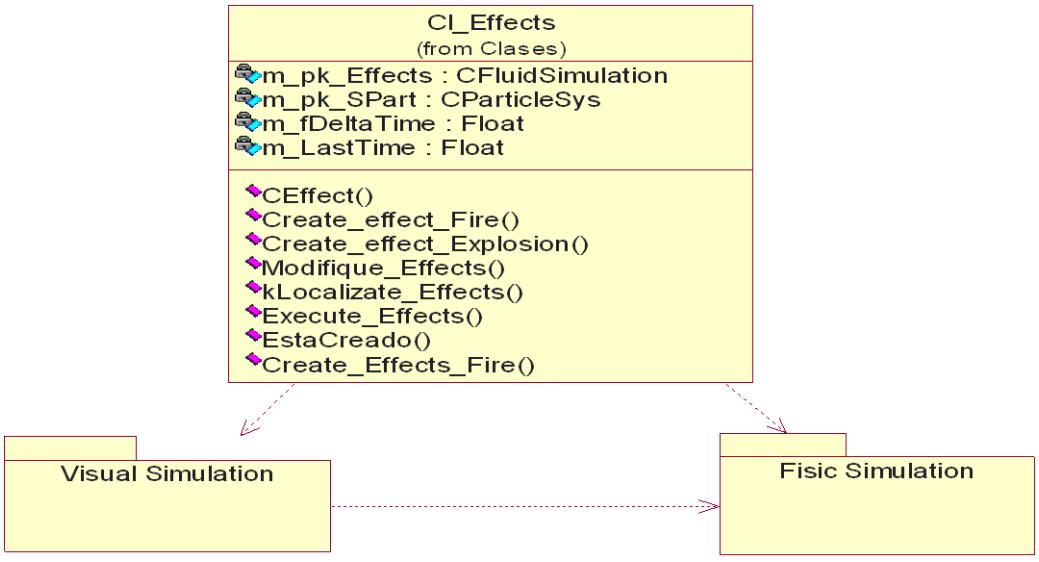

Figura 9 Diagrama de Paquetes de Clases de Diseño.

4.1.1 Paquete *"Visual Simulation"*

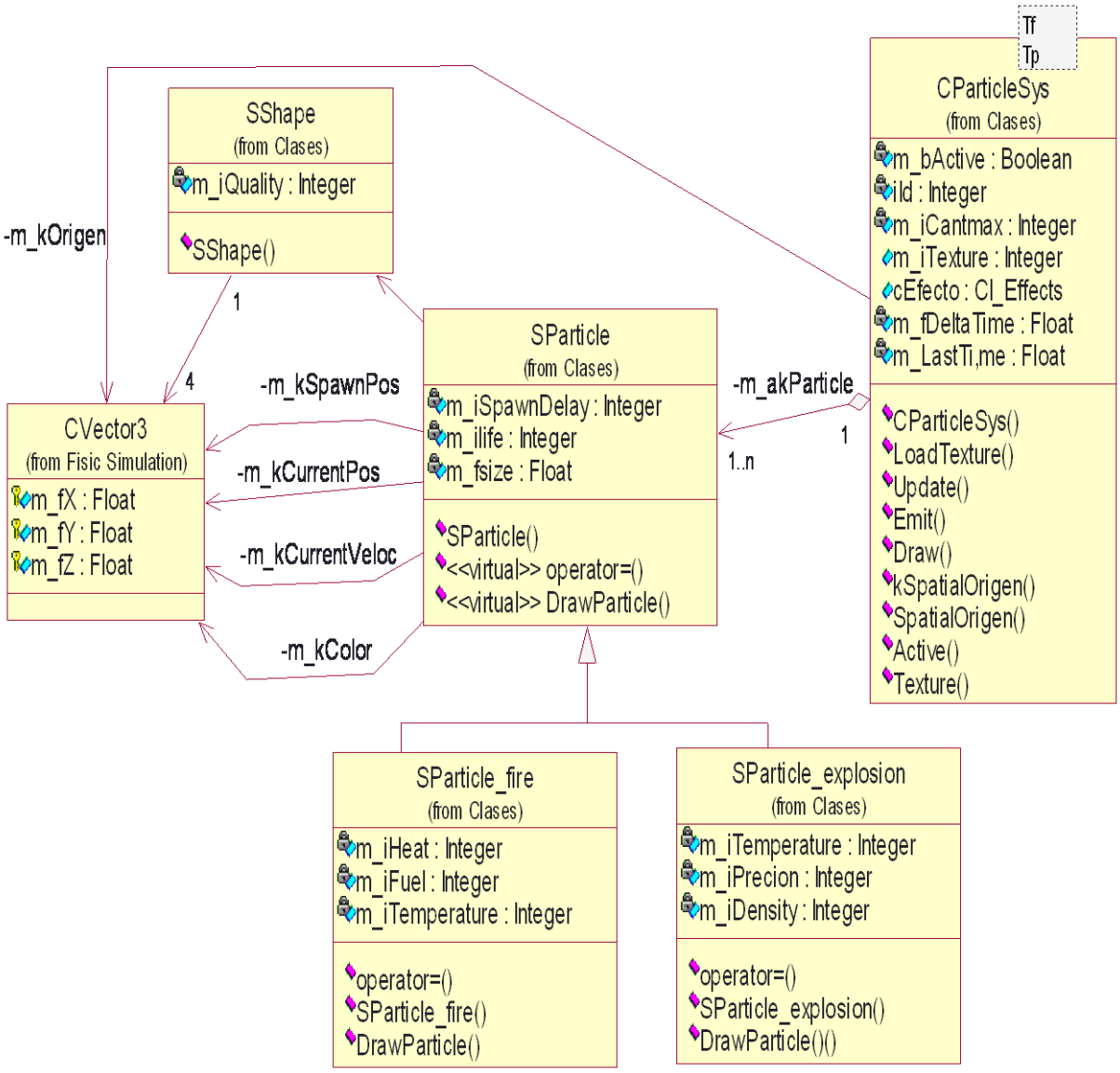

Figura 10 Diagrama de clases del paquete *"Visual Simulation"*

4.1.2 Paquete "*Fisic Simulation*"

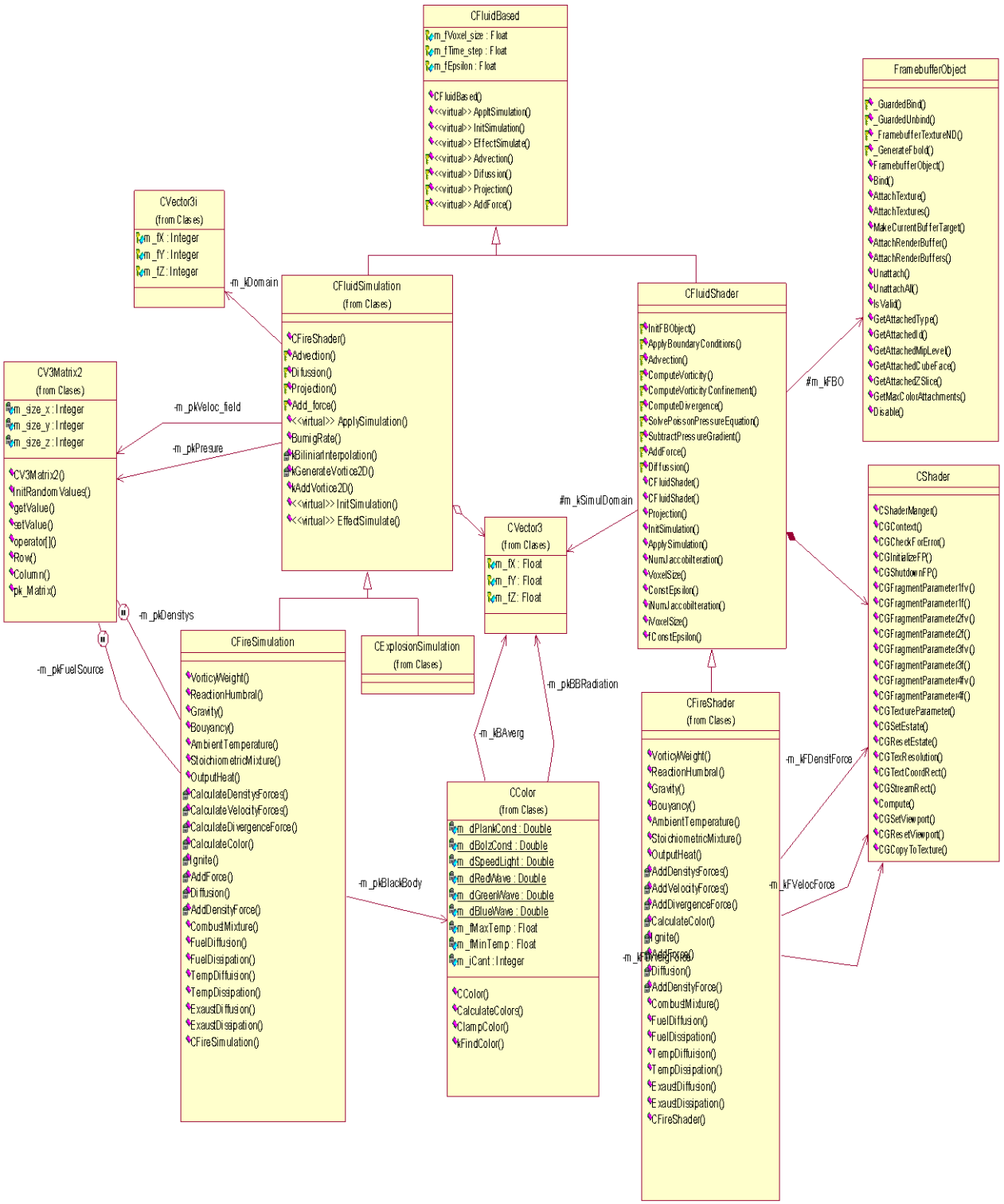

Figura 11 Diagrama de clases del paquete *"Fisic Simulation"*

4.2 Diagramas de Interacción de Diseño

4.2.2 Realización del caso de uso *"Crear efecto"*

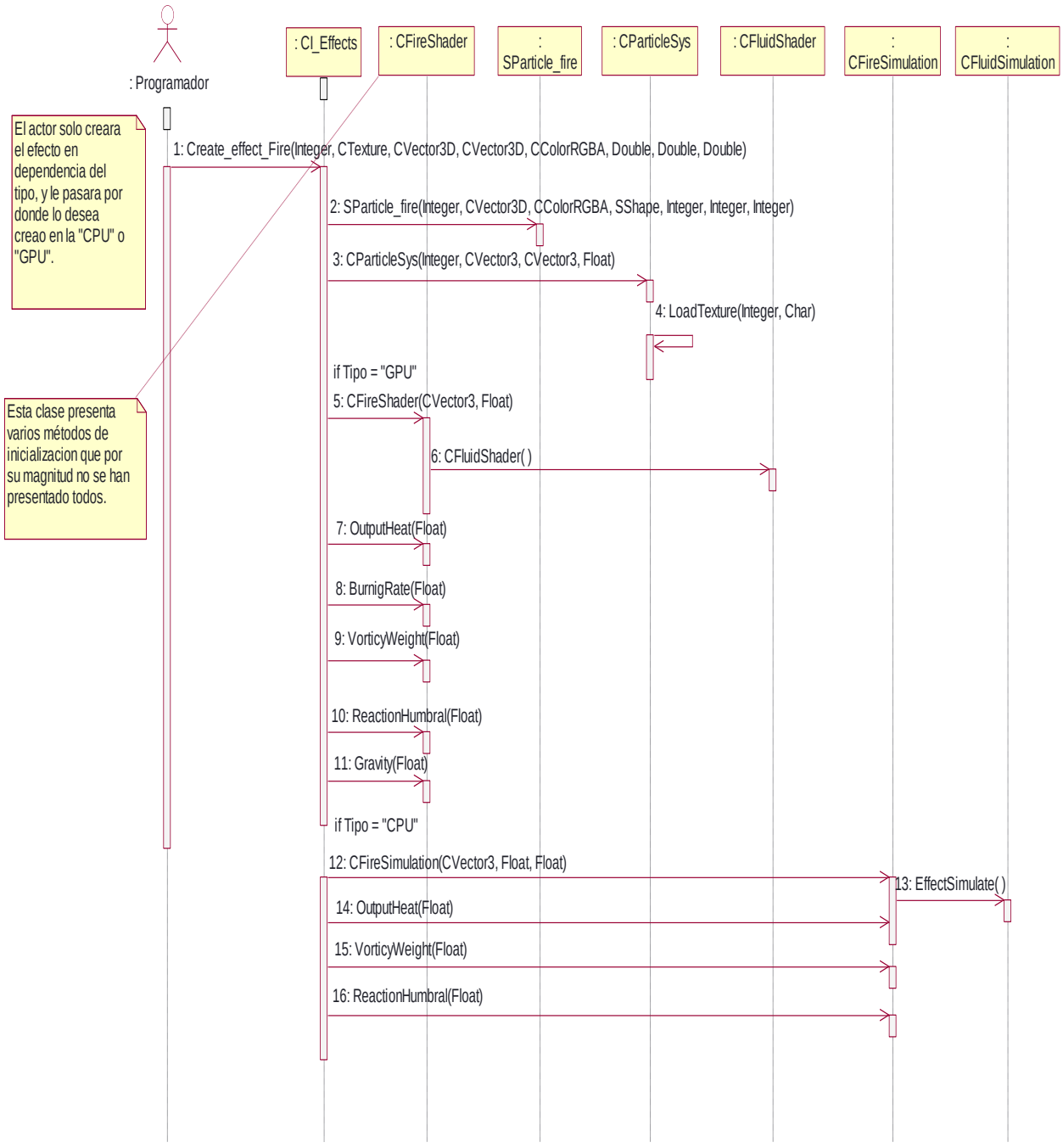

Figura 12 Diagrama de secuencia *"Crear efecto"*

4.2.3 Realización del caso de uso *"Modificar efecto"*

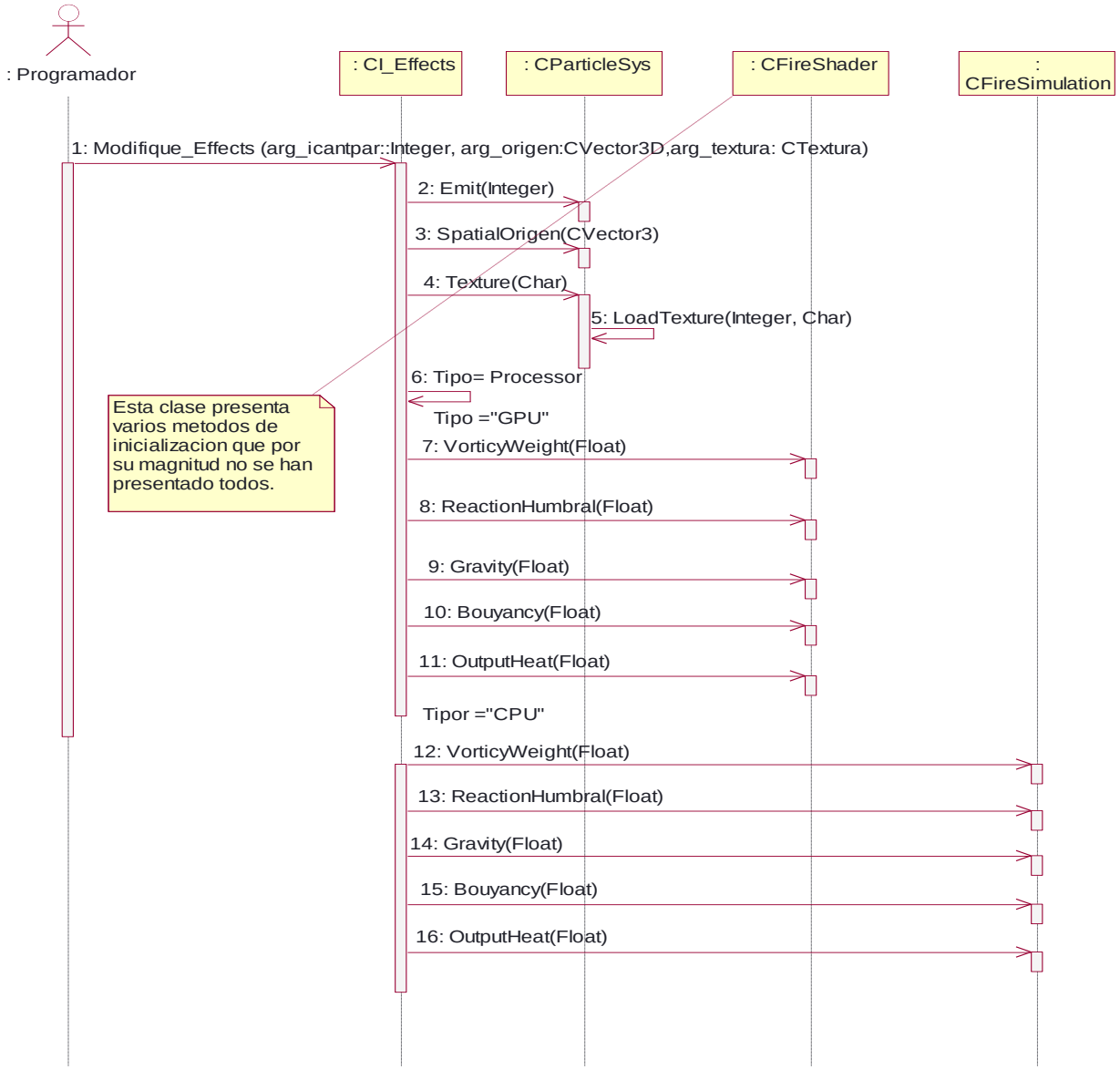

Figura 13 Diagrama de secuencia *"Modificar efecto"*

4.2.4 Realización del caso de uso *"Localizar efecto"*

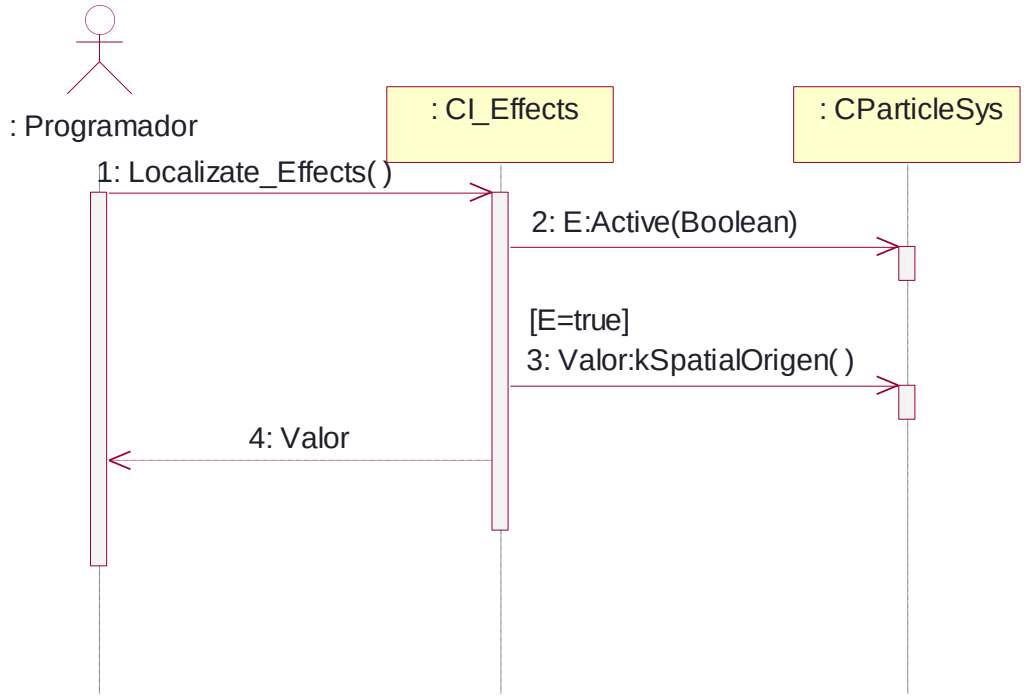

Figura 14 Diagrama de secuencia *"Localizar efecto"*

4.2.5 Realización del caso de uso *"Ejecutar efecto"*

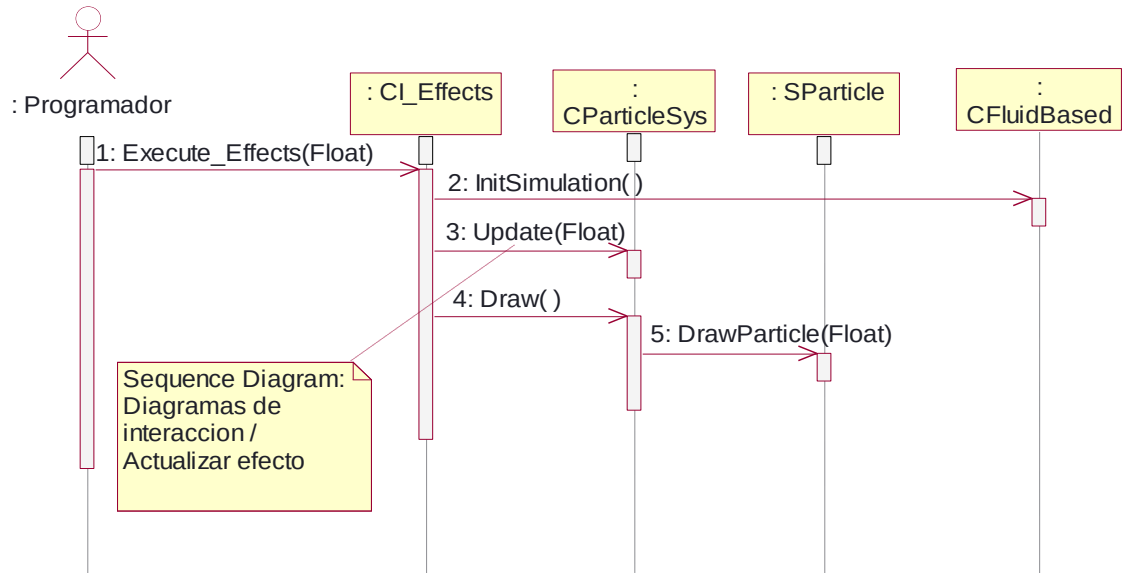

Figura 15 Diagrama de secuencia *"Ejecutar efecto"*

4.2.6 Realización del caso de uso *"Actualizar efecto"*

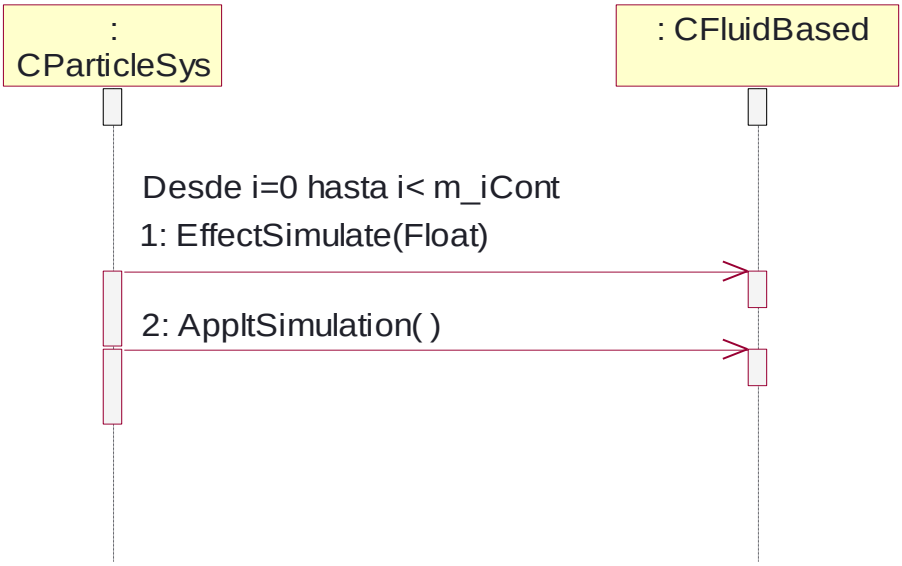

Figura 16 Diagrama de secuencia *"Actualizar efecto"*

4.3 Descripción de las clases

4.3.1 Clase Controladora" *CI_Effects"*

Tabla 9 Clase "CI_Effects"

Nombre: CI_Effects	
Tipo de clase (interfaz)	
Atributo	**Tipo**
pk_Effects	CFluidSimulation
pk_SPart	CParticleSys
Para cada responsabilidad:	
Nombre:	`Modifique_Effects(int)`
Descripción:	Modifica un efecto con los parámetros que se le entran.
Nombre:	`Execute_Effects()`
Descripción:	Ejecuta y visualiza el efecto.
Nombre:	`Localizate_Effects`
Descripción:	Localiza un efecto y devuelve la posición.
Nombre:	`CI_Effects()`
Descripción:	Constructor de la clase.

4.3.2 Paquete *"Visual Simulation"*

Tabla 10 Clase "CParticleSys"

Nombre: CParticleSys	
Tipo de clase (controladora)	
Atributo	**Tipo**
ak_Particles	Type_Particle
cant_max	int
cont	int
Texture	int
kPartDomin	CVector3
Origen_Pos	CVector3
pk_Effect	Effect_Simulation
Para cada responsabilidad:	
Nombre:	CParticleSys(int arg_icant, CVector3 arg_origen, CVector3 arg_PartDomin)
Descripción:	Constructor de la clase.
Nombre:	LoadTexture(char * arg_filename)
Descripción:	Carga una textura dada una ubicación.
Nombre:	Initialize(Type_Particle& arg_Part)
Descripción:	Inicializa la clase con los parámetros de inicio.
Nombre:	Emit(int arg_icant)
Descripción:	Emite una la cantidad de partículas pasada por parámetros.
Nombre:	Update(float arg_fDeltaTime)
Descripción	Actualiza todas las particulas.
Nombre	Draw()
Descripción	Dibuja cada particula.

Tabla 11 Clase "SParticle"

Nombre: SParticle	
Tipo de clase (entidad)	
Atributo	**Tipo**
k_SpawnPos	CVector3
k_CurrentPos	CVector3
k_CurrentVeloc	CVector3
k_Color	CVector3
k_dir	CVector3
i_SpawnDelay	int
i_life	int
f_size	float
Para cada responsabilidad:	
Nombre:	operator = (SParticle P)
Descripción:	Operador de asignación.

4.3.3 Paquete *"Fisic Simulation".*

Tabla 12 Clase "CFluidBased"

Nombre: CFluidBased	
Tipo de clase (Controladora)	
Atributo	**Tipo**
`m_fVoxel_size`	`Float`
`m_fTime_step`	`Float`
`m_fEpsilon`	`Float`
Para cada responsabilidad:	
Nombre:	`CFluidBased ()`
Descripción:	Constructor por defecto.
Nombre:	`AppltSimulation()`
Descripción:	Método virtual heredado que dado la posición y velocidad de cada partícula, las actualiza según la simulación física.
Nombre:	`InitSimulation()`
Descripción:	Método virtual heredado que dado la posición de la partícula la inicializa según la distribución de las partículas en cada efecto.
Nombre:	`EffectSimulate()`
Descripción:	Método virtual heredado que ejecuta el algoritmo de simulación física, actualiza constantemente los campos existentes (velocidad y densidad).
Nombre:	`Advection()`
Descripción:	Método que realiza la parte del auto movimiento del flujo del efecto, tanto para la velocidad como para la densidad del método de Jos Stams. Incluido con el cálculo de las condiciones del borde.
Nombre:	`Difussion()`
Descripción:	Método que realiza la parte de difusión del método de Jos Stams.
Nombre:	`Projection()`
Descripción:	Método que realiza la parte de Proyección del método de Jos Stams.
Nombre:	`AddForce()`
Descripción	Método que adiciona las fuerzas que ocacionan velocidades. Calculando las fuerza de vorticidad, fuerza de flotabilidad y la de gravedad.

Tabla 13 Clase "CFluidShader"

Nombre: `CFluidShader.`	
Tipo de clase (entidad)	
Atributo	**Tipo**
`m_fVoxel_size`	`Float`
`m_fTime_step`	`Float`
`m_fEpsilon`	`Float`
`m_kContext`	`CGcontext`
`m_kFAdvect`	`CShaderManger`
`m_kFVor`	`CShaderManger`
`m_kFVorConfin`	`CShaderManger`
`m_kFBC`	`CShaderManger`
Para cada responsabilidad:	
Nombre:	`CFluidShader()`
Descripción:	Constructor de la clase.
Nombre:	`ApplyBoundaryConditions(unsigned int fieldID, float neighborScale)`
Descripción:	Método que calcula las condiciones del borde para los campos.
Nombre:	`Advected(unsigned int velocityID, unsigned int advectedID)`
Descripción:	Método heredado y sobrecargado que realiza la parte de auto movimiento del flujo del método de Stams.
Nombre:	`ComputeVorticity(unsigned int arg_iVelocity)`
Descripción:	Calcula la vorticidad del campo de velocidad
Nombre:	`ComputeVorticityConfinement(unsigned int arg_iVorticity)`
Descripción:	Calcula la Fuerza de vorticidad a través de la vorticidad del fluido.
Nombre:	`ComputeDivergence(unsigned int arg_iVelocity)`
Descripción:	Calcula la divergencia del fluido. Es el primer paso de la parte de Proyección del método de Stams.
Nombre:	`SolvePoissonPressureEquation(unsigned int pressureID,unsigned int pressureIDaux,unsigned int`

	`divergenceID = 0,CVector3 arg_fAlpha = CVector3::ZERO, CVector3 arg_fBeta = CVector3::ZERO`
Descripción:	Método que realiza el Segunto Paso de la parte de la proyección. Halla el campo de preción haciendo uso del método iterativo de Jaccobi.
Nombre:	`InitSimulation()`
Descripción:	Método heredado y sobrecargado que dado la posición de la partícula la inicializa según la distribución de las partículas en cada efecto.
Nombre:	`Diffusion(CV3Matrix2& )`
Descripción:	Método que realiza la parte de difusión del método de Jos Stams.
Nombre:	`AddForce(unsigned int arg_iVeloc, unsigned int arg_iForces)`
Descripción:	Método que adiciona las fuerzas al campo de velocidad o de densidad.
Nombre:	`ApplySimulation()`
Descripción:	Método virtual heredado que dado la posición y velocidad de cada partícula las actualiza según la simulación física.
Nombre:	`EffectSimulate()`
Descripción:	Método virtual heredado que ejecuta el algoritmo de simulación física, actualiza constantemente los campos existentes (velocidad y densidad).

Tabla 14 Clase "CFireShader"

Nombre: CFireShader	
Tipo de clase (entidad)	
Atributo	**Tipo**
TEXTURE_SOURCE	Integer
m_kFVelocForce	CShaderManger
m_kFDensitForce	CShaderManger
m_kFDivergForce	CShaderManger
bIngnit	bool
m_fBurnigRate	Float
m_fReactionHumbral	Float
m_fGravity	Float
m_fBouyancy	Float
m_fTempAmbient	Float
m_fCombustMixture	Float
m_fOutputHeat	Float
m_fExaustDissipation	Float
m_fFuelDiffusion	Float
m_fFuelDissipation	Float
m_fTempDiffuision	Float
m_fTempDissipation	Float
m_fMassExpansion	Float
Para cada responsabilidad:	
Nombre:	CFireShader()
Descripción:	Constructor de la clase.
Nombre:	InitSimulation()
Descripción:	Método heredado y redefinido que dado la posición de la partícula la inicializa de manera inicial para el fuego.
Nombre:	EffectSimulate(float arg_fTime)

Descripción:	Método heredado y redefinido que ejecuta el algoritmo de simulación física, Actualiza constantemente los campos existentes (velocidad y densidad) para el efecto de fuego.
Nombre:	`ApplySimulation(CVector3& kPart_veloc,CVector3& kPart_pos)`
Descripción:	Método heredado y redefinido que dado la posición y velocidad de cada partícula las actualiza según la simulación física del efecto de fuego.
Nombre:	`BurnigRate(float arg_fValue)`
Descripción:	Método de entrada del valor de Ratio de quemado del combustible en la combustión.
Nombre:	`VorticyWeight(float arg_fValue)`
Descripción:	Método de entrada del valor del parámetro VorticyWeight.
Nombre:	`ReactionHumbral(float arg_fValue)`
Descripción:	Método que define el humbral de la temperatura de reacción de la combustión.
Nombre:	`Gravity(float arg_fValue)`
Descripción:	Método que define el coeficiente de gravedad.
Nombre:	`Bouyancy(float arg_fValue)`
Descripción:	Método que define el coeficiente de flotabilidad.
Nombre:	`AmbientTemperature(float arg_fValue)`
Descripción:	Método que define el la temperatura ambiente.
Nombre:	`StoichiometricMixture(float arg_fValue)`
Descripción:	Método que define el por ciento de mezcla de el combustible con el oxígeno
Nombre:	`OutputHeat(float arg_fValue)`
Descripción:	Método que define la cantidad de calor desprdedida por la combustión.
Nombre:	`CombustMixture(float arg_fValue)`
Descripción:	Método que define el coeficiente de gravedad.
Nombre:	`FuelDiffusion(float arg_fValue)`
Descripción:	Método que define el coeficiente de difusión del combustible.
Nombre:	`FuelDissipation(float arg_fValue)`
Descripción:	Método que define lel coeficiente disipación del combustible.

Nombre:	`TempDiffuision(float arg_fValue)`
Descripción:	Método que define el coeficiente de difusión de la temperatura.
Nombre:	`TempDissipation(float arg_fValue)`
Descripción:	Método que define el coeficiente de disipación de la temperatura.
Nombre:	`ExaustDiffusion(float arg_fValue)`
Descripción:	Método que define el coeficiente de difusión del gas de escape.
Nombre:	`ExaustDissipation(float arg_fValue)`
Descripción:	Método que define el coeficiente de disipación del gas de escape.

Tabla 15 Clase "CShaderManger"

Nombre: CShaderManger	
Tipo de clase (entidad)	
Atributo	**Tipo**
`m_CgContext`	`CGcontext`
`m_kfragmentProgram`	`CGprogram`
`m_ m_kprofile`	`CGprogram`
`m_fMinX`	`Float`
`m_fMinY`	`Float`
`m_fMinS`	`Float`
`m_fZ`	`Float`
`m_fResS`	`Float`
`m_fMinResS`	`Float`
`m_fTexelWidth`	`Float`
`m_fPixelWidth`	`Float`
`m_fTexelHeight`	`Float`
`m_fPixelHeight`	`Float`
`m_fResT`	`Float`
`m_fMinResT`	`Float`

m_fMaxS	Float
m_fMaxT	Float
m_fMaxY	Float

Para cada responsabilidad:

Nombre:	CShaderManger()
Descripción:	Constructor de la clase.
Nombre:	CGCheckForError(const char *arg_cSituation)
Descripción:	Método que chequea en busca de errores existentes.
Nombre:	CGInitializeFP(char* fpFilename, char* entryPoint = "",char * arg_pArg = NULL)
Descripción:	Inicializa los fragments programts (shaders).
Nombre:	CGShutdownFP()
Descripción:	Elimina los fragmentos del programa.
Nombre:	CGTexResolution(float arg_iW,float arg_iH)
Descripción:	Define la resolución de la textura (campo escalar o campo vectorial)
Nombre:	CGTextCoordRect(float arg_fMinS, float arg_fMinT, float arg_fMaxS, float arg_fMaxT)
Descripción:	Método para el manejo de las condiciones del borde y las condiciones del fluido. Da las coordenadas de texturas.
Nombre:	CGStreamRect(float arg_fMinX, float arg_fMinY, float arg_fMaxX, float arg_fMaxY,float z = 0.0f)
Descripción:	Método para el manejo de las condiciones del borde y las condiciones del fluido. Da las coordenadas en pixel.
Nombre:	Compute(bool arg_bBC = false)
Descripción:	Ejecuta el fragment program cargado.
Nombre:	CGSetViewport()
Descripción:	Define el tamaño del viewport de pintado.
Nombre:	CGResetViewport()
Descripción:	Restablece el viewport.

Tabla 16 Clase "FramebufferObject"

Nombre: `FramebufferObject`	
Tipo de clase (Entidad)	
Atributo	**Tipo**
`m_fboId`	`GLuint`
`m_savedFboId`	`GLint`
Para cada responsabilidad:	
Nombre:	`FramebufferObject ()`
Descripción:	Constructor por defecto.
Nombre:	`Bind()`
Descripción:	Habilita el frame buffer object.
Nombre:	`AttachTexture( GLenum texTarget,GLuint texId, GLenum attachment,int mipLevel,int zSlice)`
Descripción:	Adjunta una textura como FBO. Por defecto en el target GL_COLOR_ATTACHMENT0
Nombre:	`AttachTextures( int numTextures,GLenum texTarget[], GLuint texId[],GLenum attachment[],int mipLevel, int zSlice)`
Descripción:	Adjunta un conjunto de texturas como FBO en los diferentes target existentes.
Nombre:	`Disable()`
Descripción:	Desabilita todos los FBO.

Tabla 17 Clase "CFluidSimulation"

Nombre: `CFluidSimulation`	
Tipo de clase (entidad)	
Atributo	**Tipo**
`pkVeloc_field`	`CV3Matrix2`
`pkPresure`	`CV3Matrix2`
`kDomain`	`CVector3i`
`fVoxel_size`	`Float`
`fTime_step`	`Float`
`fEpsilon`	`Float`
Para cada responsabilidad:	
Nombre:	`kBiliniarInterpolation(CVector3i arg_kPos_interp, CV3Matrix2& arg_kField, CVector3 arg_kDelta_Pos = CVector3::ZERO)`
Descripción:	Método que realiza la interpolación bilineal entre las celdas vecinas. Es llamado dentro de la Advection.
Nombre:	`kGenerateVortice2D(CVector3i, CV3Matrix2&)`
Descripción:	Calcula la vorticidad del campo de velocidad.
Nombre:	`kAddVortice2D(CVector3i arg_kPos,float arg_fEpsilon)`
Descripción:	Método que calcula y retorna la fuerza de vorticidad.
Nombre:	`fDivergence(CVector3i ,CV3Matrix2&)`
Descripción:	Método que calcula la divergencia del campo. Es llamado dentro del método `kAddVortice2D`.
Nombre:	`Jaccobi2D(int,CVector3i,CVector3 arg_kAlpha ,CVector3 arg_kRBeta,CVector3 arg_kB,CV3Matrix2&)`
Descripción:	Realiza el método de Jacocobi.
Nombre:	`BoundaryCondition(CVector3i, CVector3&)`
Descripción:	Dibuja cada partícula.
Nombre:	`Advection(CV3Matrix2& )`
Descripción:	Método heredado y sobrecargado que realiza la parte de auto movimiento del flujo del método de Stams.

Nombre:	`Projection()`
Descripción:	Método que realiza la parte de la Proyección del Método de Jos Stams.
Nombre:	`Diffusion(CV3Matrix2& )`
Descripción:	Método que realiza la parte de la difusión del Metodo de Jos Stams.
Nombre:	`AddForce(CV3Matrix2&)`
Descripción:	Método que llama los métodos de kGenerateVortice2D y kAddVortice2D y adiciona la fuerza resultante al campo de velocidad.
Nombre:	`CFluidSimulation()`
Descripción:	Contructor por defecto.
Nombre:	`CFluidSimulation(CVector3 arg_kDomain,float arg_fEps, float arg_fVoxel_size)`
Descripción:	`Contructor de la clase.`
Nombre:	`VoxelSize(float)`
Descripción:	Método que define el tamaño de las celdas.
Nombre:	`ComputerDomain(CVector3)`
Descripción:	Método que define las dimensiones la regilla.
Nombre:	`VorticeScale(float)`
Descripción:	Método que define la escala de la vorticidad.
Nombre:	`TimeStep(float)`
Descripción:	Método que define el paso del tiempo.
Nombre:	`kComputerDomain()`
Descripción:	Método que retorna las dimensiones de la regilla.
Nombre:	`fVoxelSize()`
Descripción:	Método que retorna el tamaño de las celdas.
Nombre:	`ApplySimulation(CVector3& arg_kPart_veloc, CVector3& arg_kPart_pos)`
Descripción:	Método virtual heredado que dado la posición y velocidad de cada partícula las actualiza según la simulación física.
Nombre:	`EffectSimulate( float arg_fTime)`
Descripción:	Método virtual heredado que ejecuta el algoritmo de simulación física, actualiza constantemente los campos existentes (velocidad y densidad).

Tabla 18 Clase "CFireSimulation"

Nombre: `CFireSimulation`	
Tipo de clase (entidad)	
Atributo	**Tipo**
`pkDensitys`	`CV3Matrix2`
`pkFuelSource`	`CV3Matrix2`
`pkBlackBody`	`CV3Matrix2`
`fBurnigRate`	`Float`
`fReactionHumbral`	`Float`
`fGravity`	`Float`
`fBouyancy`	`Float`
`fTempAmbient`	`Float`
`fCombustMixture`	`Float`
`fOutputHeat`	`Float`
`fVorticyWeight`	`Float`
`fExaustDiffusion`	`Float`
`fExaustDissipation`	`Float`
`fFuelDiffusion`	`Float`
`fFuelDissipation`	`Float`
`fTempDiffuision`	`Float`
`fTempDissipation`	`Float`
`fMassExpansion`	`Float`
Para cada responsabilidad:	
Nombre:	`CalculateDensitysForces(CVector3i arg_kPos)`
Descripción:	Método que calcula la combustión y la disipación de las densidades de fuego.
Nombre:	`CalculateVelocityForces(CVector3i arg_kPos)`
Descripción:	Método que calcula la fuerzas de velocidades del fluido.
Nombre:	`CalculateDivergenceForce(CVector3i arg_kPos)`

Descripción:	Método que calcula las fuerzas divergentes del fluido. Expansión de masa.
Nombre:	`CalculateColor()`
Descripción:	Calcula el color correspondiente de la partícula según su temperatura.
Nombre:	`Ignite()`
Descripción:	Método que inicia le proceso de combustión del sistema. Cambia temporalmente el valor de `ReactionHumbral a -1`, para que pueda combustionar el sistema al inicio.
Nombre:	`AddVelocityForce()`
Descripción:	Adiciona el resultado de `CalculateVelocityForces` al campo de velocidad.
Nombre:	`Diffusion()`
Descripción:	Método heredado
Nombre:	`AddDensityForce()`
Descripción:	Adiciona el resultado de `CalculateDensitysForces` al campo de densidades.
Nombre:	`InitilizeFuelSource()`
Descripción:	Método que inicializa la fuente de combustible.
Nombre:	`CFireSimulation()`
Descripción:	Contructor por defecto.
Nombre:	`CFireSimulation(CVector3 arg_kDomain, float arg_fEps, float arg_fVoxel_size)`
Descripción:	`Contructor de la clase.`
Nombre:	`EffectSimulate( float arg_fTime)`
Descripción:	Método heredado y sobrecargado.
Nombre:	`ApplySimulation(CVector3& arg_kPart_veloc,CVector3& arg_kPart_pos)`
Descripción:	Método heredado y sobrecargado.
Nombre:	`VorticyWeight(float arg_fValue)`
Descripción:	Entrada de parametros.
Nombre:	`ReactionHumbral(float arg_fValue)`
Descripción:	Entrada de parámetros.

Nombre:	`Gravity(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`Bouyancy(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`AmbientTemperature(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`StoichiometricMixture(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`OutputHeat(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`CombustMixture(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`FuelDiffusion(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`FuelDissipation(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`TempDiffuision(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`TempDissipation(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`ExaustDiffusion(float arg_fValue)`
Descripción:	Entrada de parámetros.
Nombre:	`ExaustDissipation(float arg_fValue)`
Descripción:	Entrada de parámetros.

Tabla 19 Clase "CV3Matrix2"

Nombre: CV3Matrix2	
Tipo de clase (entidad)	
Atributo	**Tipo**
size_x	int
size_y	int
size_z	int
pk_Matrix	CVector3
Para cada responsabilidad:	
Nombre:	CV3Matrix2()
Descripción:	Constructor por defecto.
Nombre:	CV3Matrix2(int arg_ix, int arg_iy,int arg_iz)
Descripción:	Constructor de la clase.
Nombre:	InitRandomValues(int arg_INTERVAL)
Descripción:	Inicializa la matriz con valores random.
Nombre:	getValue(int,int,int)
Descripción:	Devuelve la posición dado tres valores pasados.
Nombre:	setValue(CVector3i,CVector3)
Descripción:	Asignar CVector3 en la posicion[x][y] de CVector3i.
Nombre:	operator[](CVector3i)
Descripción:	Retornar el valor de la posicion[x][y] del CVector3i.

Tabla 20 Clase "CVector3i"

Nombre: CVector3i	
Tipo de clase (entidad)	
Atributo	**Tipo**
m_fX	int
m_fY	int
m_fZ	int
Para cada responsabilidad	
Nombre:	CVector3i ()
Descripción:	Constructor por defecto.
Nombre:	CVector3i (int arg_fX, int arg_fY, int arg_fZ)
Descripción:	Constructor de copia.
Nombre:	CVector3i (CVector3);
Descripción:	Constructor de la clase.
Nombre:	operator* (const CVector3i& arg_kVector3)
Descripción:	Sobrecarga el operador de producto, vector*vector.
Nombre:	operator* (int arg_fFactor)
Descripción:	Sobrecarga el operador de producto, vector*int.
Nombre:	operator+ (const CVector3i& arg_kVector3)
Descripción:	Sobrecarga el operador de suma, vector+vector.
Nombre:	operator- (const CVector3i& arg_kVector3)
Descripción:	Sobrecarga el operador de diferencia, vector-vector.
Nombre:	operator == (CVector3i arg_kVector3)
Descripción:	Sobrecarga el operador de igualdad.

Tabla 21 Clase "CVector3"

Nombre: CVector3	
Tipo de clase (entidad)	
Atributo	**Tipo**
m_fX	Float
m_fY	Float
m_fZ	Float
Para cada responsabilidad:	
Nombre:	CVector3 ()
Descripción:	Constructor por defecto.
Nombre:	CVector3 (int arg_fX, int arg_fY, int arg_fZ)
Descripción:	Constructor de copia.
Nombre:	Normalize ()
Descripción:	Método de normalización.
Nombre:	operator* (const CVector3& arg_kVector3)
Descripción:	Sobrecarga el operador de producto, vector*vector.
Nombre:	operator* (float arg_fFactor)
Descripción	Sobrecarga el operador de producto, vector*int.
Nombre:	operator* (CMatrix3 arg_fMatrix3)
Descripción:	Sobrecarga el operador de producto.
Nombre:	operator* (CMatrix4 arg_fMatrix4)
Descripción:	Sobrecarga el operador de producto,con una matrix de 4.
Nombre:	operator+ (const CVector3& arg_kVector3)
Descripción:	Operador de suma, vector+vector.
Nombre:	operator- (const CVector3& arg_kVector3)
Descripción:	Sobrecarga el operador de diferencia, vector-vector.
Nombre:	operator/ (CVector3& arg_kVector3);
Descripción:	Sobrecarga el operador de división, vector/vector.

Nombre:	`operator == (CVector3 arg_kVector3)`
Descripción:	Sobrecarga el operador de de igualdad entre dos vectores.
Nombre:	`operator = (CVector3 arg_kVector3)`
Descripción:	Sobrecarga el operador de asignación.
Nombre:	`operator -= (CVector3 arg_kVector3)`
Descripción:	Sobrecarga el operador de decremento.
Nombre:	`operator += (CVector3 arg_kVector3)`
Descripción:	Sobrecarga el operador de incremento.
Nombre:	`operator < (CVector3 arg_kVector3)`
Descripción:	Sobrecarga el operador de comparación menor extricto.
Nombre:	`operator > (CVector3 arg_kVector3)`
Descripción:	Sobrecarga el operador de comparación mayor extricto.

Conclusiones

En este capítulo se confeccionaron los diagramas de clases de cada uno de los paquetes, también los diagramas de secuencias por cada CU donde se tiene una secuencia más exacta de las funcionalidades que realizan. Se realizó una descripción de las clases donde se encuentran los atributos y las operaciones más importantes.

Capítulo 5: Implementación

Introducción

Esta etapa del proyecto constituye el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .h y .cpp correspondiente a la implementación en C++. También se da a conocer según el proyecto "**Simpro**" los estándares de codificación.

5.1 Estándares de codificación

Nombre de los ficheros:

Los nombres de los ficheros .h y .cpp utilizan las iniciales del nombre de la herramienta (STK).

*ej:***STK**NameOfUnits.cpp

Constantes:

Las constantes se nombran con mayúsculas, utilizándose "_" para separar las palabras.

ej. const int MY_CONST_ZERO = 0;

Enumerados:

Para los enumerados se utiliza el indicador "**E**" en el nombre del tipo, y en las constantes se utilizan las iniciales del nombre del enumerado. Las constantes van en mayúsculas.

ej1: enum **E**MyEnum
{
ME_VALUE,

ME_OTHER_VALUE
};

ej2: enum **E**Node**T**ype
{
NT_GEOMETRYNODE,
NT_GROUPNODE...
};

Estructuras:
Se utiliza el indicador "**S**" para indicar que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.
ej: struct **S**MyStruct {…};

Clases:
Se utiliza el indicador "**C**" para indicar que es una clase. Ver más adelante la nomenclatura de las variables miembros.
ej: class **C**ClassName;

Interfaces:
Se utiliza el indicador "**I**" para indicar que es una interfaz.
ej: **I**MyInterfaceName

Listas e iteradores de la *std*:
Para los tipos de datos utilizados de la librería estándar de C++ (*vector*, *map*, *multimap*, etc.), se utiliza el indicador "**T**", con los sufijos **List**, **Map** y **MultiMap** según la estructura, así como el sufijo **Iter** para los iteradores. Además el nombre lleva el tipo de dato a almacenar en la estructura en cuestión:
*ej:*vector<CNode> **T**Node**List**;
TNodeList::iterator TNodeList**Iter**;
*ej:*map<> **T**Name**Map**;
TNameMap::iterator **T**name**MapIter**;
*ej:*multimap<> **T**Name**MultiMap**;
TNameMultiMap::iterator **T**Name**MultiMapIter**;

Declaración de variables:
Los nombres de las variables comienzan con un identificador del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepone el identificador "**m**_" (en minúscula), si son

globales se les antepone el identificador "**g**_", y en caso de ser argumentos de algún método, se les antepone el prefijo "**arg**_".

Tipos simples:

ej:

bool **b**VarName;

int **i**Name;

unsigned int **ui**Name;

float **f**Name;

char **c**Name;

char* **ac**Name; // arreglo de caracteres

char* **pc**Name; // puntero a un char

char** **aac**Name; // bidimensional

char** **apc**Name; // arreglo de punteros

bool **m_b**MemberVarName; // variable miembro de una estructura o clase

char **g_c**GlobalVarName; // variable global

short **s**Name;

void* **pv**Name;

Instancias de tipos creados:

ej:

EMyEnumerated **e**Name;

SMyStructure **k**Name;

CClassName **k**ObjectName; // objeto de una clase

CClassName* **pk**Name; // puntero a objeto

CClassName* **ak**Name; // arreglo de objetos

CClassName* **m_ak**Name; // variable miembro de clase

IMyInterface* **pI**Name; // puntero interfaces

Métodos:

En el caso de los métodos, se les antepone el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepone nada. Los constructores y destructores, como lo exigen los compiladores, llevan el nombre de la clase.

Constructor y destructor:

*ej:*CClassName (bool arg_bVarName, float& arg_fVarName);

~CClassName ();

Funciones:

ej: bool bFunction1 (…);

int* piFunction2 (…);

CClassName* pkFunction3 (…);

Procedimientos:

ej: void Procedure4 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán "Gets" ni "Sets", sino como los demás métodos, pero **con el nombre** de la variable a la que se accede y sin el prefijo "**m**_":

ej: Para la siguiente variable, los métodos de acceso serían:

int **m**_**i**MyVar; //variable

int **i**MyVar(); //"Get"

void MyVar(int arg_**i**MyVar) //"Set"

int& **i**MyVar(); //"Get" y "Set"

5.2 Diagrama de despliegue

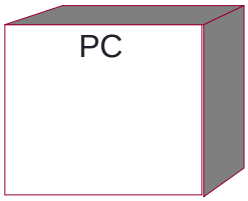

Figura 17 Diagrama de despliegue**.**

5.3 Diagramas de componentes

5.3.1 SubPaquetes de Componentes

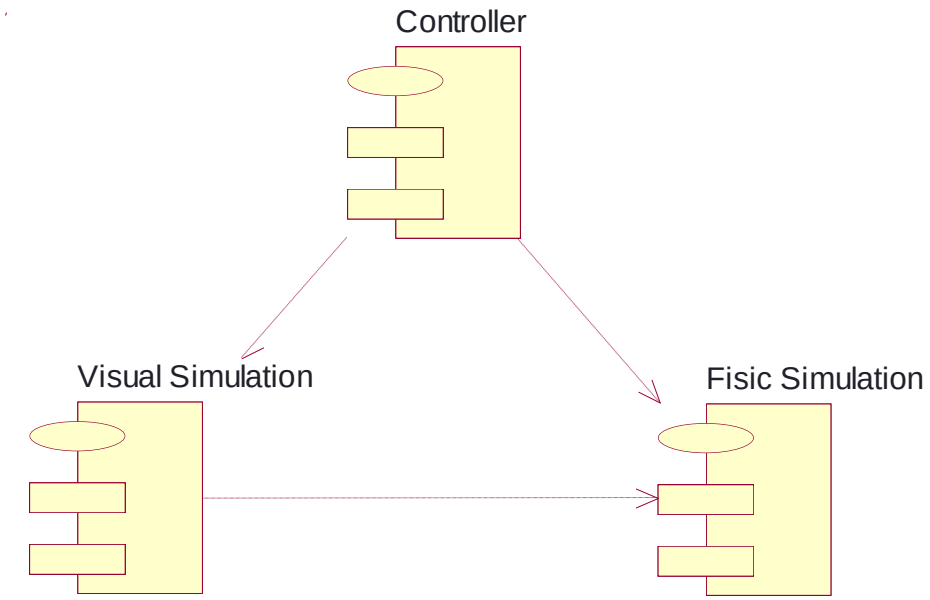

Figura 18 SubPaquetes de Componentes.

5.3.2 Diagrama de Componentes *"Controller"*

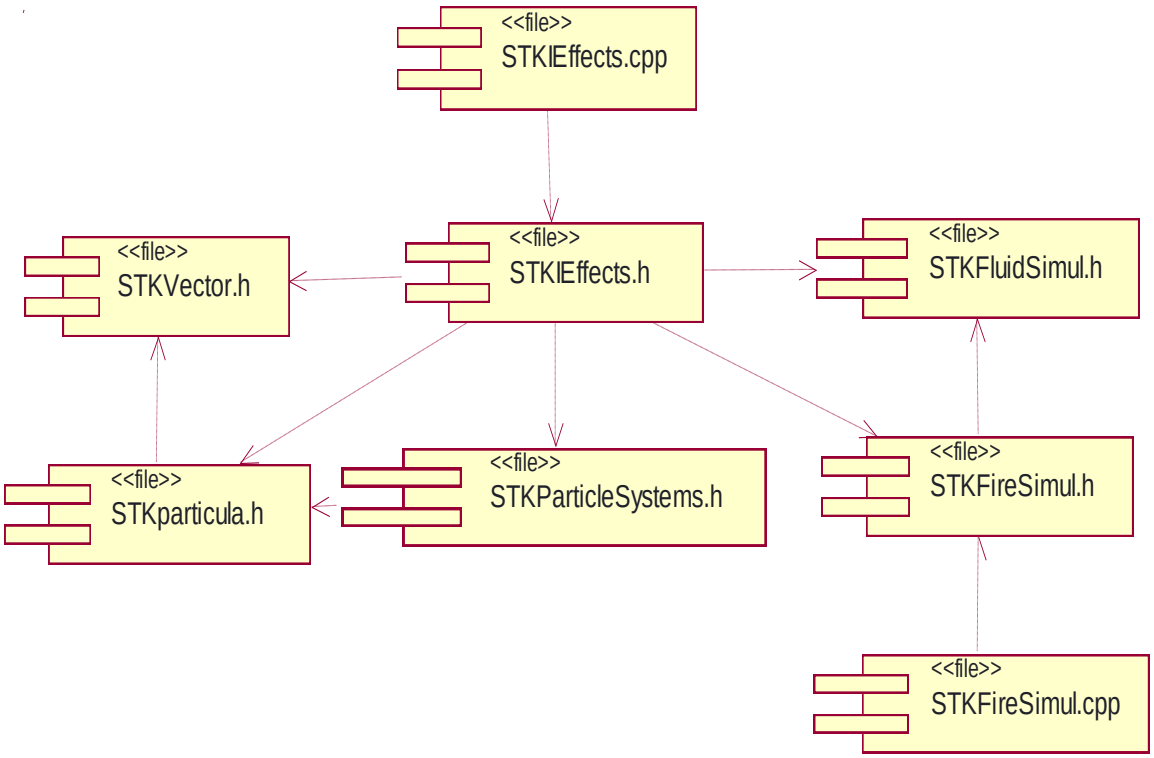

Figura 19 Diagrama de Componentes *"Controller"*

5.3.3 Diagrama de Componentes *"Fisic Simulation"*

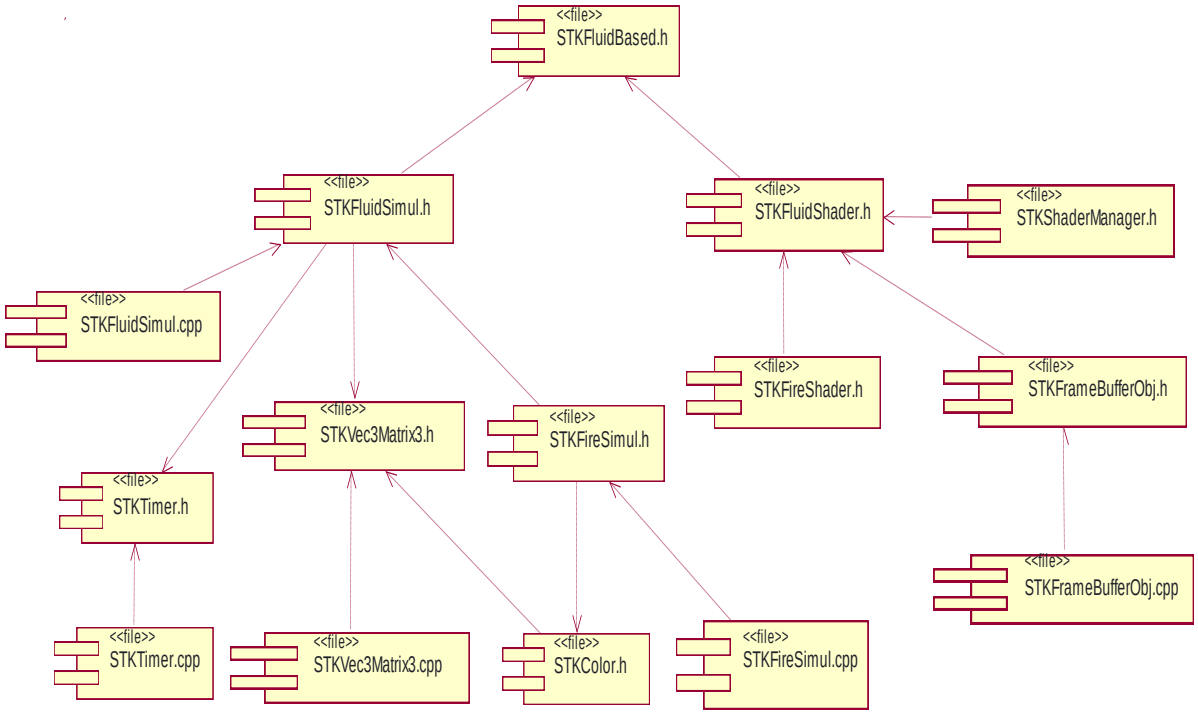

Figura 20 Diagrama de Componentes *"Fisic Simulation"*

5.3.4 Diagrama de Componentes *"Visual Simulation"*

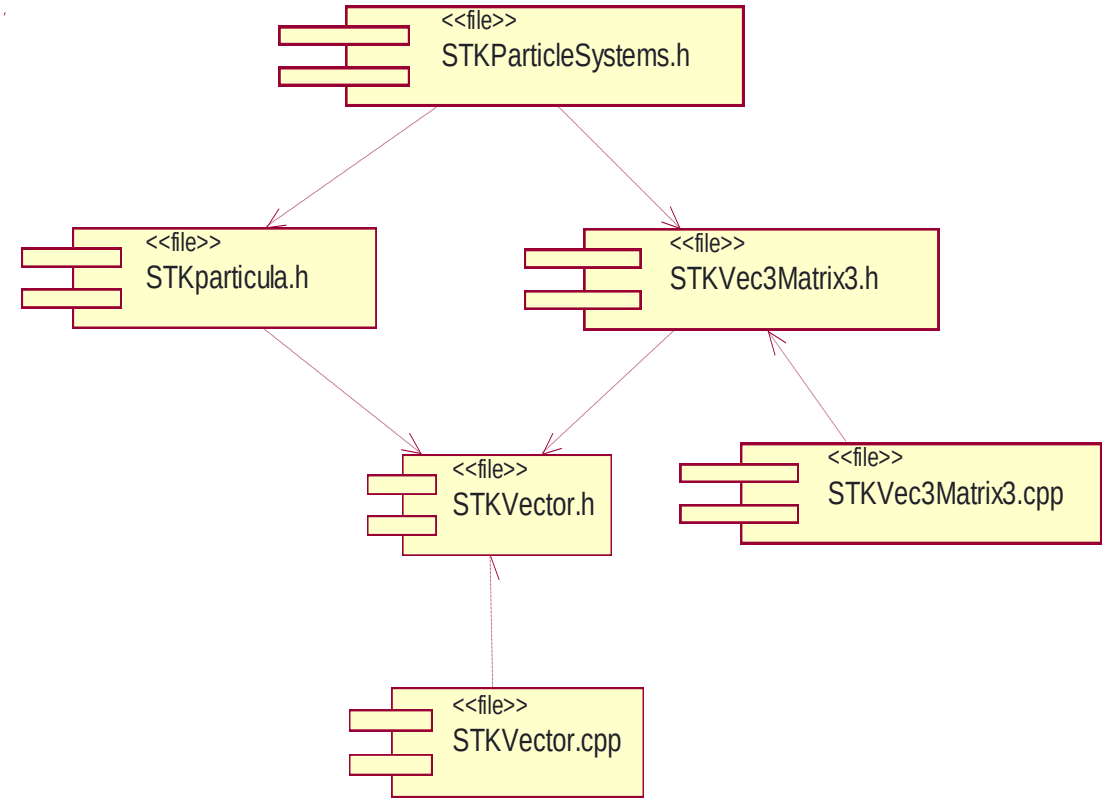

Figura 21 Diagrama de componentes *"Visual Simulation"*

Conclusiones

En este capítulo se dio a conocer los diagramas más importantes del flujo de trabajo de Implementación como son el Diagrama de despliegue y el Diagrama de componentes este último aportando, lo que viene siendo la interacción entre los .h y los .cpp. Después de logrado esto se puede pasar a la etapa de programación para el primer ciclo de vida del software.

Conclusiones

Para el cumplimiento de los objetivos y en concordancia con las exigencias hechas por la entidad cliente, se requirió el estudio de los métodos de visualización y los métodos físicos para obtener los métodos más óptimos, así como la ejecución de las tareas planteadas para el logro del proyecto.

Con el desarrollo de este trabajo se da cumplimiento a los objetivos propuesto:

- Se realizó un subsistema óptimo y extensible.
- Se elaboró un diagrama de clases el cual permite que se le agreguen otros efectos de pirotecnia.
- Se implementó y se visualizó el efecto de Fuego para una primera versión del software.

Recomendaciones

Se recomiendan los siguientes aspectos al trabajo:

- Implementar otros efectos como Explosiones y Humo.
- Vincular el subsistema realizado con la herramienta de desarrollo de Simpro.
- Lograr que los fluidos interactúen con objetos.

Referencia

1. **Storli, Geir y Rodal, Knut Erik Samuel.** Phisically Based Simulation and Visualization of Fire in Real-Time. *Norwegian University of Science and Technology,.* [En línea] 16 de 06 de 2006. [Citado el: 13 de 03 de 2007.] http://www.diva-portal.org/ntnu/undergraduate/abstract.xsql?dbid=1100.

2. **Nguyen D. Q., Fedkiw R., Jensen H. W.** Physically based modeling and animation of fire. [En línea] 2002. [Citado el: 27 de 02 de 2007.] http://graphics.ucsd.edu/~henrik/papers/fire/.

3. **Harris., Mark J.** Fast fluid dynamics simulation on the gpu. *Nvidea.* [Online] 2004. http://download.developer.nvidia.com/developer/SDK/Individual_Samples/DEMOS/OpenGL/src/gpgpu_fluid/docs/GPU_Gems_Fluids_Chapter.pdf.

4. **Mueller, Klaus, y otros.** Simulating fire with texture splats. [En línea] http://www.cs.sunysb.edu/~mueller/papers/Fire.pdf.

5. **Robert, C.D Philippe y Schweri, Daniel.** Exploring Parallelism for Real-Time Smoke Visualisation. *IAM.* [En línea] 9 de 2005. http://iamwww.unibe.ch/publikationen/techreports/2005/iam-05-003.

6. **Feldman, Bryan E., O'Brien, James F y Arikan, Okan.** Animating Suspended Particle Explosions. *Berkeley Computer Animation & Modeling.* [En línea] 2003. www.cs.berkeley.edu/b-cam/Papers/Feldman-2003-ASP.pdf.

7. **Reeves, William T.** A technique for modeling a class of fuzzy objects. *Laboratoire de Recherche en Informatique.* [En línea] 1983. www.lri.fr/~mbl/ENS/IG2/devoir2/files/docs/fuzzyParticles.pdf.

8. **Melek, Zeki y Keyser, John.** Interactive Simulation of Fire. *Department of Computer Science.* [En línea] 15 de 8 de 2002. www.cs.tamu.edu/academics/tr/tamu-cs-tr-2002-7-1.

9. **Kruger, Jens y Westermann, Rudiger.** Gpu simulation and rendering of volumetric effects for computer games and virtual environments. *Tum.3D- Computer Graphics & visualization.* [En línea] 2005. http://wwwcg.in.tum.de/Research/data/Publications/eg05.pdf.

10. **Richard, S. Bezemer.** Real Time Simulation of Natural Phenomenon Using Particle Systems and Noise Functions. [En línea] 2002.

11. http://es.wikipedia.org/wiki/Aut%C3%B3mata_celular

12**. Luis Fernando González Vargas.** Una Introducción a los Autómatas Celulares. [En Línea]

Editado el 15 de Marzo de 1999.

13. http://es.wikipedia.org/wiki/Metaball

Glosario de abreviaturas

SRV: Sistema de Realidad Virtual.

RV: Realidad Virtual.

UCI: Universidad de las ciencias informáticas.

3D: Tercera dimensión.

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

OpenGL: *Open Graphics Library* (Biblioteca de gráficos abierta).

LP: Gas Licuado de Petróleo.

CFD: Computación de la Dinámica de los Fluidos.

A.C: Autómata Celular.

GPU: Graphic Processor Unit.

UML: Lenguaje Unificado de Modelado.

CU: Caso de Uso.

Glosario de Términos

Advection[1]: Auto movimiento del fluido, movimiento propio, se interpreta como la velocidad arrastrada por la misma velocidad del fluido.

Ad hoc[5]: Significa la aplicación de medios definidos por el usuario.

Dinámica: Se refiere al estudio de la evolución temporal de los sistemas de cualquier tipo en relación a las causas que provocan o gobiernan dicha evolución [11].

Frame[2]**:** Fotograma que conforma una animación, puede ser visto como el conjunto de imágenes que juntos conforman un video.

Frame-rates[3]: Termino referente a la velocidad de visualizado de la aplicación. Relación de cantidad de frame por segundos.

Simulación: "La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con el mismo, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias -dentro de los limites impuestos por un cierto criterio o un conjunto de ellos - para el funcionamiento del sistema"[12].

Tecnología: Conjunto de teorías y de técnicas que permiten el aprovechamiento práctico del conocimiento científico.

Voxel[4]: Es una celda o cuadricula de una maya en el espacio 3D.

Renderizar[6]: Visualización grafica de objetos en el espacio 2D o 3D.

Bitmaps[7]: Mapas de bit

Lattice: En matemática, un retículo, red o lattice es un conjunto parcialmente ordenado en el cual todo subconjunto finito no vacío tiene un supremo y un íntimo. El término "retículo" viene de la forma de los diagramas de Hasse de tales órdenes.

Toroide: Es una superficie de revolución engendrada por una curva cerrada y plana que gira alrededor de una recta fija de su plano, que no la corta. De modo que un toro es un tipo particular de toroide.

Índice de figuras y tablas

Índice de figuras

Índice de tablas