UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 5 ENTORNOS VIRTUALES

Módulo para la simulación de la dinámica de los cuerpos-rígidos en entornos virtuales

Trabajo para optar por el Título de Ingeniería en Ciencias Informáticas

Autores: Liudmila Pupo Peña

Liudmila Reyes Álvarez

Tutores: Ing. Yanoski Rogelio Camacho Román

Lic. José Manuel Fernández Hechavarría

Ciudad de la Habana, mayo de 2007

"Año 48 de la Revolución"

DECLARACIÓN DE AUTORÍA

Por este medio declaramos que somos las únicas autoras de este trabajo y autorizamos al Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los __ días del mes de ___del ___.

Liudmila Pupo Peña	Liudmila Reyes Alvarez	Yanoski Rogelio Camacho Román
_____	_____	_____
Firma de la Autora	Firma de la Autora	Firma de la Tutor

DATOS DE CONTACTO

Nombre y Apellidos: Yanoski Rogelio Camacho Román

Edad: 26 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Informática

Categoría Docente: Profesor Instructor

E-mail: rcamacho@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.

Dedicatoria

Pupo: A mis padres, a mi hermano, a toda mi familia…

Reyes: A mi hija, a mis padres, a mi José Manuel y a mi abuela…

Agradecimientos

A todos los que de una forma u otra han colaborado con este trabajo o han influido en nuestras vidas como estudiantes. A la UCI, por darnos la oportunidad de cultivar el conocimiento en nuestra vida universitaria.

A nuestros compañeros de aula desde primer año, y en especial a los de este último año en la elaboración de la tesis: A Dago, Lazaro, Karel, Dasiel y Alexey por su continua preocupación por la marcha del trabajo.

Resumen

En los últimos años el desarrollo de los Sistemas de Realidad Virtual (SRV) se ha expandido considerablemente a los diferentes sectores de la sociedad. El realismo de las escenas simuladas no siempre tiene la calidad requerida, uno de los principales problemas es que no se presentan las leyes físicas elementales que rigen el comportamiento de los cuerpos. El objetivo de este trabajo es la elaboración de un módulo que permita la simulación de la dinámica de los cuerpos rígidos.

En la investigación realizada se abordan los conceptos y ecuaciones matemáticas y físicas necesarias para el soporte de la dinámica de los cuerpos, se analizan las principales técnicas para lograr la animación basada en la física y se presenta una solución al problema planteado mediante un algoritmo que permite la simulación física mediante métodos de resolución de ecuaciones diferenciales, destacando el método de Rungue Kutta.

El módulo resultante de este proyecto, desarrollado usando el proceso unificado de software e implementado en c++ estándar, está acoplado a la herramienta “Scene ToolKIt”, permitiendo un mayor realismo en los productos finales que se elaboren, logrando incorporar leyes físicas que determinan el comportamiento de los cuerpos rígidos, según factores externos (fuerzas) que incidan sobre él.

Palabras Claves

Realidad Virtual, simulación basada en física, dinámica de cuerpos rígidos

Contenido

Introducción

A finales de los 80´, los gráficos por computadora entraron en una nueva época. No era que las soluciones tridimensionales (3D) comenzaran a reemplazar los enfoques bidimensionales y de dibujos de líneas (2D), sino que también existía la necesidad de un espacio de trabajo totalmente interactivo generado a través de la tecnología.

Es así como se presenta la Realidad Virtual(RV), un modelo matemático que describe un “espacio tridimensional”, dentro del cual están contenidos objetos que pueden representar desde una simple entidad geométrica hasta una forma sumamente compleja como puede ser un desarrollo arquitectónico[1].

La RV tiene múltiples aplicaciones: la investigación genética y química (estructura del átomo, hélice del ADN), la exploración espacial (futura estación espacial Alpha o la programación de un recorrido en Marte), en la medicina (quirófano virtual), en el entretenimiento (juegos), en la meteorología. También se benefician todas las profesiones que utilizan habitualmente maquetas o prototipos, ya que la RV permite al diseñador presentar a sus clientes simulaciones tridimensionales de la obra antes de iniciar su realización. Las actividades de alto riesgo constituyen otro campo donde la RV se puede convertir en un aliado invaluable, permitiendo, entre otras cosas, el entrenamiento de personal especializado sin poner en peligro su integridad física.

En una gran cantidad de aplicaciones es objetivo fundamental que el mundo virtual se aproxime lo mejor posible al mundo real, es por eso que surge la necesidad de incorporar a los Sistemas de Realidad Virtual (SRV) modelos físicos, siendo esta una de las ramas fundamentales en el desarrollo de Entornos Virtuales.

Es el momento que Cuba, que está inmersa en un programa revolucionario de informatización, dé un salto en el tema de la animación por computadoras a un nivel que le permita ocupar un puesto en el creciente mercado de la simulación.

Ya se han dado los primeros pasos, existe el “Centro de Investigación y Desarrollo de Simuladores” (SIMPRO), pionero en el desarrollo de este campo en nuestro país, y ha surgido también como perfil investigativo y de producción de la Facultad 5, en la Universidad de la Ciencias Informática (UCI), donde ya se perciben logros en esta rama tan amplia de la informática, como es el desarrollo de un simulador de conducción de auto para la exportación.

La Facultad 5 de la UCI cuenta con un grupo de proyectos tanto investigativos como de producción encargados del desarrollo de SRV. Para facilitar el trabajo de estos proyectos, es conveniente contar con módulos preelaborados para el trabajo con redes, ejecución de sonidos, manipulación de objetos del entorno, inteligencia artificial, modelación física-matemática, etc., elementos que se integran en una herramienta básica para Sistemas de Realidad Virtual de nombre “Scene ToolKit” producida por uno de los proyectos de la Facultad 5 de la UCI, para el uso común de todos los demás proyectos.

La herramienta “Scene ToolKit” carece de un módulo que soporte la dinámica de los cuerpos y que permita que estos interactúen en la escena siguiendo las ecuaciones que rigen su movimiento y se vean afectadas por factores externos(fuerzas) que modifiquen en el tiempo la posición y orientación del cuerpo, siendo esto una **necesidad** importante del proyecto.

Por tanto el **problema** a resolver queda formulado a modo de interrogante de la siguiente forma: ¿Cómo lograr la incorporación de la dinámica de los cuerpos rígidos a los entornos virtuales que se desarrollen con la herramienta Scene ToolKit en los proyectos de la Facultad 5 de la UCI?

El **objeto de estudio** de este trabajo lo constituye la dinámica de los cuerpos rígidos en los simuladores y juegos virtuales **campo de acción** que abarca este trabajo es la incorporación de la dinámica de los cuerpos rígidos a la herramienta “Scene ToolKit” desarrollada en la Facultad 5 de la UCI.

El **objetivo general** de esta investigación es desarrollar un módulo que incorpore la simulación de la dinámica de los cuerpos-rígidos a los Sistemas de Realidad Virtual desarrollados por los proyectos de la Facultad 5 de la Universidad de las Ciencias Informáticas mediante la herramienta “Scene ToolKit”.

Se plantean entonces un grupo de tareas que permitirán satisfacer el objetivo, y que se pueden resumir en las siguientes:

- Investigar las leyes y/o ecuaciones de la dinámica de los cuerpos rígidos en su estado de movimiento.
- Investigar algoritmos y técnicas actuales utilizados para simular la dinámica de los cuerpos en los SRV.
- Estudiar la herramienta “Scene ToolKit” de la que va a formar parte el módulo a crear.
- Seleccionar la Metodología de Análisis y Diseño que facilite la creación y garantice la calidad del módulo.
- Seleccionar las herramientas para llevar a cabo el proyecto y la elección de la plataforma en la que se desarrollará la aplicación.
- Implementar un módulo que permita dar solución al objetivo propuesto.

El desarrollo de este módulo debe realizarse con el fin de que sea de fácil uso por los programadores de las aplicaciones finales, y que sea flexible a actualizaciones futuras, es decir, que se le puedan añadir nuevas leyes físico-matemáticas sin tener que transformar la arquitectura básica del funcionamiento del módulo.

El contenido de este trabajo se encuentra distribuido de la siguiente forma:

En un primer capítulo, “Fundamentación Teórica”, se hace un análisis bibliográfico donde se investigan las características de los SRV, las leyes físico-matemáticas elementales que rigen a los objetos en movimiento y las técnicas, algoritmos y tendencias actuales para simular la dinámica de los cuerpos rígidos en los juegos y simuladores actuales. En el capítulo 2, “Características del Sistema”, se exponen las características técnicas que presentará el sistema como solución al problema planteado, se analiza con mayor profundidad el objeto de estudio, y se crea el modelo del dominio, se hace la captura de requisitos y se crean los modelos de casos de uso del sistema. En el capítulo 3, “Diseño e Implementación del sistema”, se presentan los diagramas de clases, interacción y componentes. Finalmente, se ofrece un glosario de abreviaturas y uno de términos para ayudar a la comprensión del lenguaje técnico utilizado a lo largo del trabajo.

1

Capítulo 1 Fundamentación Teórica

Introducción

El movimiento de los objetos en el mundo creado debe producir la impresión de integración física y movilidad desde el punto de vista de un espacio en tres dimensiones. [2] Alcanzar un modelo físico-matemático eficiente que logre cumplir con todos los requisitos que el sistema necesite, y funcione de forma que los objetos interactúen y alcancen la movilidad de la forma más real posible, exige seleccionar de forma minuciosa los algoritmos y métodos que se deben seguir para modelar el mundo físico con tanta precisión como sea posible.

En este capítulo se hace un estudio de las leyes físicas y ecuaciones de la dinámica que rigen el movimiento de los cuerpos, así como sus propiedades. Se analizan algoritmos y tendencias actuales de cómo se desarrolla este tema en el mundo y se presentan algunos motores físicos que se utilizan para juegos y simuladores en la actualidad.

1.1 Animación Generada por Ordenador

La forma más útil de clasificar la multitud de técnicas de animación computacional usadas en los gráficos por computadora es según el nivel de abstracción con que el animador especifica el movimiento [3]. Podemos encontrar las Técnicas de Bajo Nivel en las que hay un control preciso de la posición del objeto, orientación y velocidad, y en el segundo caso las Técnicas de Alto Nivel, en las que se utilizan algoritmos complejos que establecen el comportamiento, movimiento o respuesta de los objetos (Simulación Física). [4]

En las Técnicas de Bajo Nivel se controla de forma precisa la posición del objeto y la cámara a lo largo del tiempo: trayectorias, velocidad y aceleración [5]. Se requiere que el animador especifique manualmente cada uno de los parámetros del movimiento individual [3]. Estas técnicas se basan principalmente en el principio de la interpolación, es decir construcción de una función continua tomando valores determinados en correspondencia con argumentos fijos previamente especificados. [5]

Dados $x_0, x_1, ..., x_n$ (abscisas) obtener una función $f(x)$ tal que $F(x_i) = y_i$ donde $y_0, y_1, ..., y_n$ son también datos. Una misma trayectoria puede recorrerse de diferentes maneras y con distintas velocidades y aceleraciones. Es necesario separar la geometría (forma de la curva) del movimiento (velocidad/aceleración). [5]

El caso de las Técnicas de Alto Nivel es la animación basada en la física, a continuación se detallan algunos aspectos de vital importancia para la comprensión de cómo funciona este tipo de animación y que aspectos físicos se deben de tener en cuenta para establecer el comportamiento, interacción y movimiento de los objetos.

1.1.1 Animación Física

La animación física se basa en la aplicación de modelos físicos de la realidad para definir la evolución de los elementos animados de la escena. [6] Los objetos se mueven de acuerdo con leyes físicas: gravedad, elasticidad, fluidez y por tanto reaccionan a estas leyes.

En cualquier escena se tendrán disímiles cuerpos para simular su movimiento y estos también pueden interactuar entre sí. Pueden ser desde autos, tanques, árboles hasta un buen número de situaciones o fenómenos naturales como: humo, nieve, polvo, tejidos, fuego o hasta sistemas planetarios.

Todos estos cuerpos en la literatura de los gráficos por computadora pueden ser clasificados en dos categorías fundamentales, los cuerpos deformables y los cuerpos rígidos. No obstante del tipo de objeto, el movimiento de estos generado por la simulación física se basa en la dinámica. Sin embargo la simulación de los cuerpos deformables o cuerpos no rígidos consume mucho más tiempo que la de los cuerpos rígidos. [7]

Se entiende por **sólido rígido** o **cuerpo rígido** a un conjunto de puntos del espacio que se mueven de tal manera que no se alteran las distancias entre ellos sea cual sea la fuerza actuante. [8] Los cuerpos rígidos no se deforman, están totalmente rígidos en su estructura y permanecen así aunque otro cuerpo interactúe con él, también se asume que no existe penetración en la interacción aunque si puede rebotar en respuesta a la colisión con otros cuerpos. [9]

Los cuerpos **no rígidos** son aquellos que sufren una deformación (cambio del tamaño o la forma) debido a la aplicación de una o más fuerzas sobre él. [10]

La animación tradicional a menudo rompe las leyes de la Naturaleza, y suele definir movimientos atractivos, pero imposibles en la realidad. Para realizar animaciones realistas, hay que tener en cuenta esas leyes de la Naturaleza: animación basada en leyes físicas, que utiliza la cinemática y la dinámica. Muchos movimientos cotidianos son muy difíciles de reproducir. [11]

La mecánica clásica es la rama de la física que estudia el reposo y la dinámica de los cuerpos rígidos; la **dinámica** se divide en cinemática y cinética, la **cinemática** estudia el movimiento de los cuerpos rígidos sin considerar las fuerzas que lo propician o influyen, en la **cinética** esencialmente se realiza el análisis del movimiento de un cuerpo real, considerado que teóricamente tal cuerpo no se deforma y estableciendo las fuerzas que: propician (motriz), contrarrestan o producen (en) el movimiento del cuerpo. [12]

La dinámica es la base de la Simulación Física [13], se estudian aquí las fuerzas que actúan sobre los objetos y sus reacciones a ellas.

La dinámica de las partículas y los cuerpos rígidos tienen determinados conceptos y modelos físicos que rigen su comportamiento. Un cuerpo presenta determinadas propiedades como posición en el espacio, velocidad lineal y angular, centro de masa, fuerzas y torques, momento lineal y angular, tensor de inercia, entre otras, todas estas propiedades se deben tener en cuenta para lograr una simulación eficiente. [14]

Utilizando la dinámica en los cuerpos rígidos es posible obtener gran realismo en la escena, pero resulta difícil especificar la animación. Hay que tomar en consideración masas, aceleraciones, grados de libertad, restricciones al movimiento, movimientos prioritarios, etc. La dinámica de los cuerpos rígidos articulados es más sencilla que la de los cuerpos deformables. Se distingue:

Dinámica directa: a partir de las masas y fuerzas aplicadas, se calculan las aceleraciones. [11]

Dinámica inversa: a partir de las masas y aceleraciones, se calculan las fuerzas que hay que aplicar. [11]

1.1.2 Bucle de animación física

El proceso de control de animación utilizando física consiste en la resolución de un conjunto de ecuaciones lineales que responden a los siguientes elementos generales: [6]

- Determinar las constantes dinámicas del sistema.
- Evaluar las condiciones iniciales.
- Encontrar las fuerzas y sus puntos de aplicación.
- Calcular las resultantes de las fuerzas y momentos.
- Resolver las ecuaciones de movimiento.

Las diferentes fórmulas involucradas en los pasos anteriores son explicadas en los epígrafes 1.2 y 1.3 donde se hace un análisis de las diferentes variables que intervienen en la simulación física y como se obtienen cada una de ellas.

1.2 Conceptos Matemáticos

1.2.1 Sistemas de Coordenadas

Para representar la posición de una partícula en el espacio es necesario establecer un sistema de coordenada. Si se desea describir, medir y analizar las características geométricas de los objetos hay que colocarlos dentro un marco de referencia, en este caso, un sistema de coordenadas.[15] Se centrará el estudio a dos sistemas:

Sistema de Coordenadas Global o del mundo: Este es el sistema de coordenadas primario (también llamado sistema de coordenadas del mundo), siendo sus tres ejes *x, y, z*. Es en este marco de referencia en el cual se definen y se posicionan todos los objetos geométricos que se encuentran en el entorno tridimensional. Además se refieren todos los objetos con respecto a este.[15]

Sistema de Coordenadas Local: Se utiliza un sistema de coordenadas local (también llamado sistema de coordenadas del modelo) para definir a un objeto independientemente del sistema de coordenadas global, sin tener que especificar una posición u orientación con respecto al sistema de coordenadas global.[15]

1.2.2 Representación de Rotaciones

Las formas más comunes de representar rotaciones son:[16]

- Ángulos de Euler.
- *Quaternion*.
- Matrices.

1.2.2.1 Ángulos de Euler

Los ángulos de Euler [16] son los que se miden respecto de cada uno de los ejes x, y, z. Especificando los tres ángulos (α, β, γ) se puede representar cualquier rotación. Esta es la forma más simple en principio, pero es incómoda en muchas situaciones.

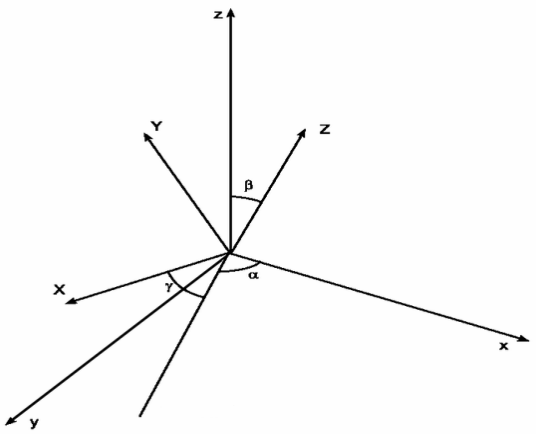


Figura 1 Rotación mediante ángulos de euler

1.2.2.2 Quaternions

Los *quaternions* extienden el concepto de rotación en tres dimensiones a rotación en cuatro dimensiones. Usando *quaternions* se puede rotar un objeto sobre el vector (x, y, z) y un ángulo de rotación *theta,* donde $w = \cos(theta\ /\ 2)$. Las operaciones con *quaternions* son computacionalmente más eficientes que la multiplicación de matrices de $4x4$ usadas para las transformaciones y las rotaciones.[17]

Los quaternions adicionan un cuarto elemento en los valores de (x, y, z) que definen un vector, resultando un vector de cuatro componentes. Sin embargo, las siguientes fórmulas ilustran como cada elemento del vector que define el quaternions se relaciona con el eje y el ángulo de rotación, donde q representa una unidad quaternions, (x, y, z, w) el eje normalizado y theta es la rotación en contra de las manecillas del reloj alrededor del eje de rotación.[17]

q.x = sin (theta/2) * axis.x q.y = sin (theta/2) * axis.y

q.z = sin (theta/2) * axis.z q.w = cos (theta/2)

La ventaja de los *quaternions* reside en que sólo usan 4 valores, por lo cual existe mucha menos redundancia en los datos, lo que produce menos errores. Además, se puede detectar si un *quaternion* ya no es correcto por medio de su módulo, que debe valer uno. Cuando esto no ocurre, se procede a normalizarlo y el problema está solucionado. Otra ventaja de los *quaternions* es que en otras aplicaciones donde se necesita interpolar rotaciones, los *quaternions* son más útiles y tienen menos error en sus operaciones.[16]

1.2.2.3 Matrices de Transformación

Las matrices es otra de las formas de representar las rotaciones. Se realiza respecto a un eje principal coordenado donde α es el ángulo a rotar.

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \quad R_y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \quad R_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figura 2 Matrices de rotaciones

El problema es que una matriz de 3X3 guarda 9 números, más que los tres ángulos de Euler o los cuatro que usan los *quaternions*.

Cuando se realiza una simulación, las matrices se están modificando todo el tiempo, esto hace que al tiempo de estar corriendo la simulación, la matriz empieza a acumular errores y pierde coherencia de modo que ya no describe una rotación solamente, sino también una transformación distinta que tiende a "deformar" el espacio. [16]

1.3 Magnitudes Físicas

A continuación se presentarán las principales magnitudes físicas que rigen la dinámica de las partículas y los cuerpos rígidos. Estas leyes físicas son de vital importancia para simular el comportamiento de un cuerpo en un entorno determinado.

1.3.1 Posición de una partícula

Se denota $X(t)$ como la posición de la partícula en el espacio del mundo (el espacio que todas las partículas y cuerpos están ocupando en el transcurso de la simulación) en el tiempo t. [18]

1.3.2 Velocidad Lineal y Angular

La **velocidad lineal** $V(t)$ de una partícula se obtiene por medio de la siguiente derivada: $V(t) = X(t)'$

X y V son vectores, por lo tanto V será un vector cuyos componentes son los de X derivados respecto al tiempo. Se le llama **rapidez** al módulo de la velocidad, que representa las unidades o metros por segundo que recorrería la partícula si esta rapidez se mantuviera constante.[16]

La **velocidad angular** se representa por el vector W , que apunta en dirección al eje de rotación del cuerpo, y cuyo módulo es a lo que se le llama rapidez angular. Esta **rapidez angular** es el desplazamiento angular del cuerpo en un período de tiempo en que esta rapidez se mantiene constante. Considerando un cuerpo en rotación como se muestra en la figura, donde $X(t)$ es la posición del centro de masa del cuerpo, $V(t)$ es la velocidad del centro de masa y $W(t)$ indica el eje y rapidez de rotación del cuerpo.[16]

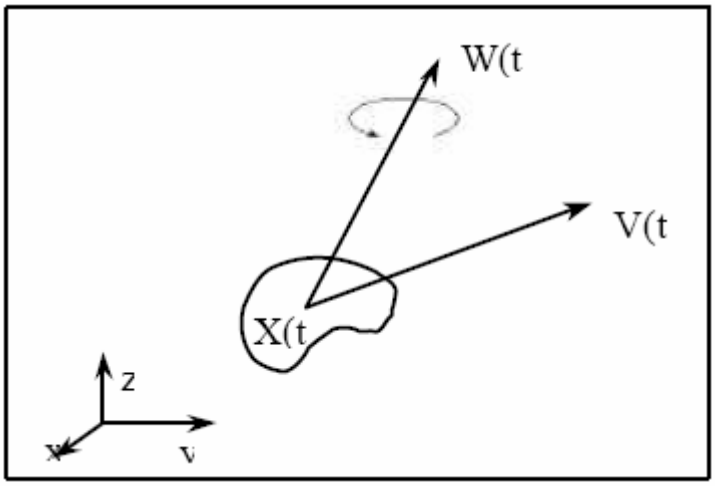


Figura 3 Representación de las velocidades linear y angular de una partícula

En el caso de la velocidad lineal siempre se calcula derivando el vector posición respecto al tiempo, pero la relación entre la velocidad angular y la rotación depende de como se represente esa rotación. Si se utilizan los *quaternions* para las rotaciones, es necesario establecer una relación entre la velocidad angular y este *quaternion*. Esta relación esta dada por:

$Q(t) = 1/2W(t) * q(t)$ [16]

Para poder multiplicar $W(t)$ y $q(t)$, es necesario que el primero se exprese como el *quaternion* $(0, W(t))$ y no como un vector $3x1$. [16]

Si se usan matrices para las rotaciones la relación esta dada por: $R(t) = W(t) * R(t)$

1.3.3 Velocidad de una partícula

Si se define una partícula o punto cuya posición en coordenadas globales es $X_p(t)$ y en coordenadas locales relativas a su centro de masa es $X_{rp}(t)$. Y suponiendo que esta partícula forma parte de un cuerpo cuyo centro de masa se encuentra en $X(t)$ y que tiene una velocidad lineal de $V(t)$ y una velocidad angular $W(t)$. [16]

La relación entre $X_p(t)$ y $X_{rp}(t)$ será la siguiente:

$$X_p(t) = X(t) + X_{rp}(t)$$

La derivada de $X_p(t)$ puede expresarse en función de las velocidades como:

$$X_p(t)' = W(t)x[X_p(t) - X(t)] + V(t)$$

Esta ecuación separa la velocidad en dos componentes, un componente lineal que viene dado por $V(t)$ y un componente angular dado por $W(t)x[X_p(t) - X(t)]$. [16]

1.3.4 Centro de gravedad

Debido a que un cuerpo es una distribución continua de masa, en cada una de sus partes actúa la fuerza de gravedad. El **centro de gravedad** es la posición donde se puede considerar actuando la fuerza de gravedad neta, es el punto ubicado en la posición promedio donde se concentra el peso total del cuerpo. Para un objeto simétrico homogéneo, el centro de gravedad se encuentra en el centro geométrico, pero no para un objeto irregular. [16]

1.3.5 Centro de masa

La definición del centro de masa permite separar la dinámica de los cuerpos en la parte lineal y la parte angular. El centro de masa de un cuerpo en coordenadas globales se define como: [18]

$$CM = \frac{\sum m_i X_i(t)}{m}$$

En la simulación se supone que un cuerpo está formado por un conjunto de partículas muy pequeñas, cada una con una masa m_i y una posición X_{ri} (en coordenadas locales). De esta forma, el centro de masa de un cuerpo en coordenadas locales se define como:

$$CM_r = \frac{\sum m_i X_{ri}(t)}{m}$$

Donde m es la masa total, o sea la suma de las masas de cada una de las partículas:

$$m = \sum m_i$$

Asumiendo que las coordenadas locales de un cuerpo tienen su origen en el centro de masa, el centro de masa de un cuerpo expresado en sus coordenadas locales se encuentra en la posición (0, 0, 0).

1.3.6 Fuerzas y Torques

Cuando se imagina una fuerza actuando sobre un cuerpo rígido es debido a varias influencias externas (ej. gravedad, aire, fuerzas de contacto), y que estas fuerzas están actuando sobre una partícula en particular del cuerpo.

Se define entonces la fuerza con los siguientes atributos: [16]

- Módulo.
- Dirección.
- Punto de aplicación.

El módulo de la fuerza indica qué cantidad de aceleración puede provocar sobre un cuerpo dado en la dirección de esta fuerza. El punto de contacto es donde se aplica la fuerza, que se puede imaginar que es una partícula muy chica que integra el cuerpo. [18]

En la simulación, en el momento que se desea mover un objeto de posición el primer paso es deducir todas las fuerzas que están actuando sobre este objeto. La suma (o el total) de todas esas fuerzas es lo que determina cómo el objeto se mueve. Algunas de las fuerzas son obvias: un enemigo perfora o golpea a jugador con el pie, el jugador empuja un objeto grande, se lanza una bola, o se lanza un misil. Sin embargo, hay algunas fuerzas menos obvias que no pueden ser pasadas por alto como: gravedad, fricción, resistencia del viento (si es significativa), o la fuerza normal. Se analizará cómo trabaja cada una de estas fuerzas, comenzando con la gravedad.

La aceleración debido a la **gravedad**, se representa en la mayoría de las fórmulas con la constante g.

Siempre que se miran los componentes verticales del movimiento, se utiliza $-g$ o $-9.8m/s^2$ para la aceleración. Ahora se puede utilizar esa constante gravitacional para calcular la fuerza debido a la gravedad, también conocida como peso. El **peso** es realmente una fuerza, que quiere decir un vector, y su dirección está siempre hacia abajo, hacia el centro de la tierra.

$W = m \cdot g$ Donde m= masa y g= aceleración debido a la gravedad ($-9.8m/s^2$ en la Tierra).

Mientras que la mayoría de los juegos no necesitarán distinguir entre la masa y el peso, en juegos extremadamente detallados (o cualquier simulación) tal distinción es necesaria. El peso de un objeto cambia si la aceleración de la gravedad cambia. Por ejemplo, la gravedad de la luna es 1/6 más fuerte que la de la tierra, cualquier peso de un objeto va a ser 1/6 de que es en la tierra. Sin embargo la masa siempre debe permanecer como estaba antes, no tiene importancia donde esté el objeto.

La diferencia más grande entre la masa y el peso es que la masa es una cantidad escalar, y el peso es una cantidad vectorial. La masa es a menudo cuantificada en gramos o kilogramos, mientras que el peso es cuantificado en libras o newton.

La fuerza **normal** es la fuerza que es aplicada por la tierra o cualquier superficie a un objeto para prevenir que este se mueva, esta fuerza se le aplica perpendicular a la superficie y opuesta a la fuerza de gravedad, así estas dos fuerzas se pueden cancelar. [19]

Otro tipo de fuerza que a menudo se ignora es la fuerza debido a la **fricción**. En los juegos a menudo se considera ignorar la fricción, pero eso depende completamente del programador. Sin embargo, si se está intentando codificar un simulador absolutamente realista, podría ser sumamente importante pasarse los ciclos del reloj extras en buscar la fricción interesada. Todo depende de cómo se quiere equilibrar la velocidad contra el realismo.

La fricción depende de dos factores: la superficie por donde se está resbalando y el objeto que está resbalando. La mayoría de los modelos físicos sólo tienen en cuenta el primer factor, pero en la realidad, los dos cuentan. [19]

Hay dos tipos de fricción realmente: la estática y cinética. La fricción estática es la fuerza que impide un objeto moverse inicialmente, y la fricción cinética es la fuerza que reduce la velocidad de un objeto después de que este consigue el movimiento. Siempre es necesario calcular la fricción estática primero. Si todas las otras fuerzas sumaran menos que la fricción estática, el objeto no se moverá. En el momento que las otras fuerzas son mayores que la fricción estática, el objeto empieza a moverse, y la fricción cinética toma valor.

La letra griega (miu) es el símbolo estándar para el coeficiente de fricción. Con este coeficiente se pueden calcular ambos tipos de fricción:

Fricción estática: $F_e = -\mu_e N$ Fricción cinética: $F_c = -\mu_c N$

Donde N es la fuerza normal.

Después que se tienen todas las fuerzas que actúan sobre un objeto, el próximo paso es sumarlas todos para encontrar la fuerza neta o total.

Cuando una fuerza es aplicada provoca una fuerza equivalente sobre el centro de masa que es responsable de la aceleración lineal resultante del cuerpo y un **torque** que sería una especie de "fuerza angular" que provoca una aceleración angular en el cuerpo. [16]

Si se define $F_i(t)$ como la fuerza total que actúa sobre una partícula i-ésima del cuerpo, $X(t)$ como la posición del centro de masa del cuerpo en coordenadas globales y $X_i(t)$ como la posición de la i-ésima partícula en coordenadas globales, se define el torque que esta fuerza produce sobre el cuerpo como:

$$T_i(t) = [X_i(t) - X(t)]xF_i(t)$$

En la siguiente figura se aprecia como actúa el torque sobre una partícula que se le ha aplicado una fuerza. El torque difiere de la fuerza en que depende la posición $X_i(t)$ de la partícula, relativa al centro de masa $X(t)$. [18]

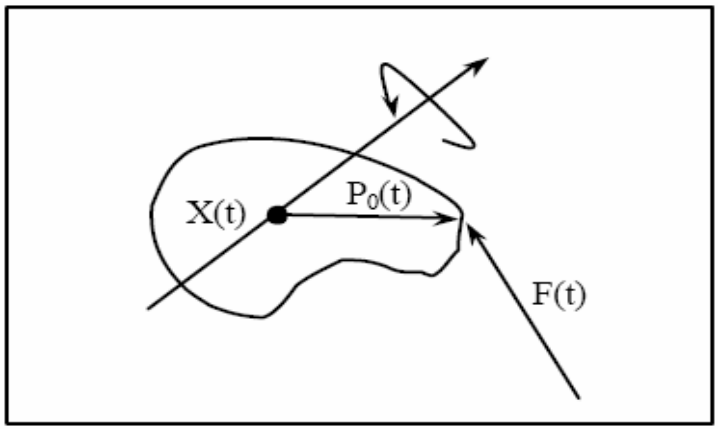


Figura 4 Torque sobre una partícula que se le aplica una fuerza

La dirección del torque $T_i(t)$ es la del eje sobre el cual el cuerpo rotará debido a éste, que es igual a la de la velocidad angular $W_i(t)$ que este producirá.

La fuerza total externa $F(t)$ actuando sobre un cuerpo es la suma de las fuerzas $F_i(t)$:

$$F(t) = \sum F_i(t)$$

Donde el torque total externo es definido similarmente como: [18]

$$T(t) = \sum T_i(t) = \sum (X_i(t) - X(t)) \times F_i(t)$$

Una propiedad interesante del torque es que depende del punto de aplicación de la fuerza, pero la fuerza equivalente sobre el centro de masa es siempre la misma.

1.3.7 Leyes de Newton

Las leyes de Newton dicen cómo la fuerza neta afecta el movimiento de un objeto.

La **Primera ley de Newton**, conocida también como Ley de inercia, dice que si sobre un cuerpo no actúa ningún otro, este permanecerá indefinidamente moviéndose en línea recta con velocidad constante (incluido el estado de reposo, que equivale a velocidad cero). [20]

La Primera Ley de Newton especifica lo que pasa cuando la fuerza neta o total en un objeto es 0. Si todas las fuerzas individuales se cancelan, no debe haber ningún cambio en el movimiento.

Si F_{net} = 0, aquí no hay cambio en el movimiento.

La **Segunda ley de Newton** se encarga de cuantificar el concepto de fuerza. Esta ley dice que la fuerza neta aplicada sobre un cuerpo es proporcional a la aceleración que adquiere dicho cuerpo por su masa. Tanto la fuerza como la aceleración, son magnitudes vectoriales, es decir, tienen además de un valor, una dirección y un sentido. De esta manera, la Segunda ley de Newton debe expresarse como:

$$\vec{F_{net}} = m\vec{a}$$

La unidad de fuerza en el Sistema Internacional es el **Newton** y se representa por **N**. Un Newton es la fuerza que hay que ejercer sobre un cuerpo de un kilogramo de masa para que adquiera una aceleración de 1 m/s^2, o sea: 1 N = 1 Kg · 1 m/s^2 [20]

Como se nota cuando la fuerza neta es 0, la aceleración debe ser 0 que es exactamente lo que la Primera Ley de Newton dice.

La **Tercera ley de Newton**, también conocida como "Principio de acción y reacción" dice que si un cuerpo A ejerce una acción sobre otro cuerpo B, éste realiza sobre A otra acción igual y de sentido contrario.

Esto es algo que se puede comprobar a diario en numerosas ocasiones. Por ejemplo, cuando se quiere dar un salto hacia arriba, empujamos el suelo para impulsarnos. La reacción del suelo es la que nos hace saltar hacia arriba. Hay que destacar que, aunque los pares de acción y reacción tengan el mismo valor y sentidos contrarios, no se anulan entre sí, puesto que actúan sobre cuerpos distintos. [20]

1.3.8 Momento Lineal

El momento lineal o cantidad de movimiento p de una partícula con masa m y velocidad V se define como:

$$P = mV$$

El momento lineal total $P(t)$ para un cuerpo rígido es la suma del producto de la masa y la velocidad de cada partícula: [18]

$$P(t) = \sum m_i . X_i(t)'$$

La velocidad de $X_i\ (t)'$ de la partícula i-ésima se define como:

$$X_i(t)' = W(t) \cdot [X_i(t) - X(t)] + V(t)$$

De esta manera el momento lineal de un cuerpo es: [18]

$$P(t) = \sum m_i . X_i(t)' = \sum (m_i V(t) + m_i W(t) \cdot (X_i(t) - X(t))) = \sum m_i V(t) + W(t) \cdot \sum m_i (X_i(t) - X(t))$$

Si se toma al centro de masa como el origen de coordenadas, es decir, si $X_i(t)$ está en coordenadas locales, la última sumatoria es cero (equivale al centro de masa en coordenadas locales) y se obtiene: [16]

$$P(t) = (\sum m_i).V(t) = M.V(t)$$

Esto da el resultado de que el momento linear total de un cuerpo rígido es igual a como si el cuerpo fuera simplemente una partícula de masa M y velocidad $V(t)$. Debido a esto, se tiene una transformación simple entre $P(t)$ y $V(t)$, siendo M una constante: [18]

$$V(t)' = \frac{P(t)'}{M}$$

Además, de acuerdo a las leyes de Newton, la fuerza que actúa sobre el centro de masa de un cuerpo es igual a la derivada de la cantidad de movimiento: [16]

$$F(t) = P(t)'$$

1.3.9 Momento Angular

El momento angular de un cuerpo se define como:

$$L(t) = I(t).W(t)$$

Donde $W(t)$ es la velocidad angular del cuerpo y $I(t)$ es una matriz de 3X3 llamada tensor de inercia. Esta última representa la inercia rotacional del cuerpo, y depende de cómo esté distribuida la masa del cuerpo respecto del centro de masa. [16]

El tensor de inercia $I(t)$ depende de la orientación del cuerpo pero no depende de la traslación. Hay que tener en cuenta para los casos lineal y angular, que el ímpetu o momento es una función lineal de velocidad, en el caso angular el factor de posicionamiento es una matriz, mientras que en el caso lineal es simplemente un escalar. El momento angular $L(t)$ es independiente de cualquier efecto de traslación, mientras que el momento lineal $P(t)$ es independiente de cualquier efecto rotatorio. [18]

El concepto de momento angular de un cuerpo no es para nada intuitivo, pero es muy útil ya que se conserva en la naturaleza. Para un cuerpo al que no se le aplican fuerzas, su momento angular se mantiene constante, mientras que la velocidad angular podría cambiar (esto sucede si cambia el momento de inercia en el tiempo). [16]

La relación entre el momento angular y el torque aplicado es: [18]

$$T(t) = L(t)'$$

Que es análogo a la relación que se establece entre la fuerza aplicada en el centro de masa y el momento lineal que se definió anteriormente.

1.3.10 Tensor de Inercia

Para cuerpos simples y que rotan alrededor de un eje fijo, el tensor de inercia se calcula analíticamente mediante integrales. Pero para las simulaciones esto no se aplica ya que los valores de la matriz dependen del eje de rotación, que cambia todo el tiempo al aplicar torques arbitrarios sobre los cuerpos. [16]

La definición del tensor de inercia contiene integrales que se deben expresar como sumatorias que se obtienen al considerar el cuerpo un conjunto de partículas con masas muy pequeñas. Si $X_{ri}(t)$ es la posición de la partícula i-ésima del cuerpo expresada en coordenadas locales (relativa al centro de masa) y m_i su masa, el tensor de inercia se define como: [16]

$$I(t) = \sum \begin{pmatrix} m_i(Xr_{iy}^{\ 2} + Xr_{iz}^{\ 2}) & -m_i Xr_{ix}\ Xr_{iy} & -m_i Xr_{ix}\ Xr_{iz} \\ -m_i Xr_{iy}\ Xr_{ix} & m_i(Xr_{ix}^{\ 2} + Xr_{iz}^{\ 2}) & -m_i Xr_{iy}\ Xr_{iz} \\ -m_i Xr_{iz}\ Xr_{ix} & -m_i Xr_{iz}\ Xr_{iy} & m_i(Xr_{ix}^{\ 2} + Xr_{iy}^{\ 2}) \end{pmatrix}$$

Figura 5 Tensor de inercia expresado como una sumatoria

Donde $X_{rix}, X_{riy}, X_{riz}$ son los componentes de X_{ri}

Como puede verse, si un cuerpo está rotando las posiciones X_{ri} de las partículas dependen de la rotación, por lo tanto habría que calcular $I(t)$ cada vez que un cuerpo rota en alguna dirección. Esto es muy pesado y haría la simulación tan lenta que sería impensado que funcione en tiempo real. La solución se encuentra intentando separar una parte del cálculo asumiendo una rotación específica, y luego recalcular el tensor de inercia para cualquier otra rotación. [16]

Si *Id* es la matriz identidad 3X3 y usando la propiedad:

$$Xr_i^{\ T} \cdot Xr_i = Xr_{ix}^{\ 2} + Xr_{iy}^{\ 2} + Xr_{iz}^{\ 2}$$

El tensor de inercia puede escribirse como:

$$I(t) = \sum m_i([Xr_i^{\ T} \cdot Xr_i] \cdot Id - Xr_i \cdot Xr_i^{\ T})$$

Si se supone que el cuerpo tiene una rotación inicial dada, la posición de las partículas que las constituyen pueden expresarse en función de su posición inicial. Entonces si se considera a $X_i(t)$ como un vector $X_{0i}(t)$ afectado por la rotación descrita por la matriz $R(t)$, se tiene que: $X_{ri} = X_{r0i}(t) + R(t)$. Aquí, $X_{r0i}(t)$ es la posición inicial de la partícula i-ésima. [16]

Reemplazando en la expresión anterior de $I(t)$ y reagrupando se llega a:

$$I(t) = R(t) \cdot \sum m_i([Xr_{0i}{}^T \cdot Xr_{0i}] \cdot Id - Xr_{0i} \cdot Xr_{0i}{}^T) \cdot R(t)^T$$

Y si se define a $I_0(t)$ como la matriz: $I_0(t) = \sum m_i([Xr_{0i}{}^T \cdot Xr_{0i}] \cdot Id - Xr_{0i} \cdot Xr_{0i}{}^T)$

Entonces el tensor de inercia queda definido por la siguiente expresión: [16]

$$I(t) = R(t) \cdot I_0 \cdot R(t)^T$$

Como los $X_{0i}(t)$ no dependen de la rotación instantánea del cuerpo, I_0 tampoco ya que queda en función de matrices constantes. Por lo tanto, $L(t)$ ha quedado definido por una parte invariable que puede ser precalculada antes de la simulación (I_0) multiplicada por derecha e izquierda por $R(t)$ y $R(t)^T$, que sí dependen del tiempo. Esto permite que el tensor de inercia sea calculado de forma más eficiente ya que este cálculo sólo involucra dos multiplicaciones de matrices, de manera tal que el trabajo más pesado se realiza antes de la simulación y se guarda en la matriz I_0. [16]

1.3.10.1 Diagonalización del Tensor de Inercia

El tensor de inercia es una matriz simétrica, y como tal diagonalizable, esto es, reducible a la forma diagonal eligiendo convenientemente los ejes de referencia:

En forma diagonal quedaría de la siguiente forma:

$$I(t) = \sum \begin{pmatrix} m_i(Xr_{iy}{}^2 + Xr_{iz}{}^2) & 0 & 0 \\ 0 & m_i(Xr_{ix}{}^2 + Xr_{iz}{}^2) & 0 \\ 0 & 0 & m_i(Xr_{ix}{}^2 + Xr_{iy}{}^2) \end{pmatrix}$$

Figura 6 Tensor de inercia diagonalizado

Existen, por tanto, en el movimiento del sólido rígido, unos ejes de referencia con origen en el Centro de Masas y que tienen direcciones tales que el Tensor de Inercia adquiere la forma diagonal. Estos ejes se

denominan Ejes Principales de Inercia y las direcciones de los mismos, Direcciones Principales de Inercia. Las componentes no nulas del tensor, que forman la diagonal del mismo, son los Momentos Principales de Inercia. [21]

Los momentos principales de inercia tienen que ver, fundamentalmente, con la simetría del sólido. Si el sólido admite una cierta simetría se determinan sus ejes principales de forma sencilla puesto que el centro de inercia y las direcciones de estos ejes mantienen la misma simetría. [21]

La ventaja de la diagonalización de la matriz del Tensor de Inercia es evidente, pues reduce el costo de cómputo de dicha matriz en un 66 %, es decir, sólo se realizan un tercio de los cálculos que serian necesarios sin la diagonalización. [21]

En la modelación de los cuerpos rígidos para la simplificación de los cálculos asociados al tensor de inercia se utilizan diferentes sólidos homogéneos cuyas matrices del Tensor de Inercia diagonalizadas son conocidas y se encuentran tabuladas en el apéndice de fórmulas.

1.4 Interacción entre cuerpos

En el epígrafe anterior se analizaron las magnitudes físicas a tener en cuenta para el análisis de una partícula o de un sistema de estas. Otro aspecto importante es analizar el momento en que los cuerpos de la escena colisionan entre sí. El proceso de colisión se puede dividir en dos etapas:

- ✓ Detección de la colisión.
- ✓ Respuesta a la colisión.

La detección de la colisión debe determinar si dos cuerpos están colisionando para luego analizar si es necesario aplicar el impulso correspondiente. La información a obtener de la intersección de dos cuerpos es el punto de colisión y la normal de colisión. El punto de colisión será útil para analizar si los cuerpos están colisionando o si se están separando. La normal de colisión sirve para determinar la dirección del impulso a aplicar. [16]

La detección de la colisión no forma parte del objetivo de esta investigación, es por eso que sólo se brinda una pequeña información de cómo ocurre, sin entrar en detalles.

La respuesta a la colisión entre los cuerpos, es la etapa en que se intentan separar los cuerpos que colisionaron, para esto se aplica el método del impulso, consiste en aplicar una fuerza producto del choque a los cuerpos que colisionaron que tiende a separarlos. A esta fuerza que es muy grande y se aplica por un período de tiempo muy chico, se le asocia un impulso, de forma que a los cuerpos les causa un cambio de velocidad instantánea. Este impulso se calcula en función de la velocidad, de las masas de los cuerpos y del coeficiente de restitución.

1.4.1 Impulso

Un impulso J es una fuerza muy grande que se aplica por un período de tiempo muy chico: $\boldsymbol{J} = \boldsymbol{F} \bullet \Delta \boldsymbol{t}$
Este impulso se aplica en dirección a la normal de colisión y provoca un cambio instantáneo en la cantidad de movimiento de $\Delta \boldsymbol{P} = \boldsymbol{J}$. [16]

El torque impulsivo que genera J será $T_J = \left[C - X(t)\right] \otimes J$, donde C es el vector del punto de colisión y $X(t)$ es el vector de coordenadas del centro de masa del cuerpo. [16]

Este torque impulsivo generará un cambio instantáneo en el momento angular de $\Delta \boldsymbol{L} = \boldsymbol{T_J}$.

Como el impulso debe separar los cuerpos que colisionaron, deberá ser opuesto para los cuerpos a y b en cuestión:

$\boldsymbol{J_a} = +\boldsymbol{j} \bullet \boldsymbol{N} \qquad \boldsymbol{J_b} = -\boldsymbol{j} \bullet \boldsymbol{N}$

Donde j es el valor modular del Impulso y N el vector normal al plano de colisión y se obtiene:

$$\boldsymbol{N} = -\Delta \boldsymbol{t} \bullet \left(\dot{\boldsymbol{C}}_{\boldsymbol{b}} - \dot{\boldsymbol{C}}_{\boldsymbol{a}} \right)$$

Y

$$\dot{C}_a(t_0) = V_a(t_0) + W_a(t_0) \times [C_a(t_0) - X_a(t_0)]$$

$$\dot{C}_b(t_0) = V_b(t_0) + W_b(t_0) \times [C_b(t_0) - X_b(t_0)]$$

Donde $W_a(t)$ y $W_b(t)$ son las velocidades angulares de los cuerpos a y b respectivamente. [16]

Para obtener el valor del módulo de J hay que comparar la velocidad relativa antes y después del choque. Las leyes que rigen las colisiones hacen uso de un coeficiente llamado coeficiente de restitución que es un factor de escala entre ambas velocidades:

$$\boldsymbol{V}_{rel}^{+} = -\boldsymbol{\xi} \bullet \boldsymbol{V}_{rel}^{-}$$

Donde ξ es el coeficiente de restitución, $\boldsymbol{V}_{rel}^{-}$ y $\boldsymbol{V}_{rel}^{+}$ son las velocidades relativas de los cuerpos antes y después de la colisión. [16]

El valor de ξ esta entre 0 y 1(inclusive) y establece cuanta energía se pierde en el choque. Si ξ=1, no se pierde energía (colisión elástica) y si ξ=0, los cuerpos pierden toda velocidad en la dirección normal (colisión instantánea). [16]

Las relaciones entre las velocidades antes (V_a^-) y después (V_a^+) del choque son:

$$V_a^{+}(t_0) = V_a^{-}(t_0) + \frac{j \cdot N(t_0)}{M_a}$$

$$W_a^{+}(t_0) = W_a^{-}(t_0) + I_a^{-1}(t_0) \cdot \{[C_a(t_0) - X_a(t_0)] \times j \cdot N(t_0)\}$$

Donde M_a es la masa del cuerpo a y I_a^{-1} es la inversa de su tensor de inercia. Combinando estas dos ecuaciones y comparando con la de C_a^{+} se obtiene una relación entre las velocidades relativas antes y después de la colisión:

$$\dot{C}_a^{+}(t_0) = V_a^{+}(t_0) + W_a^{+}(t_0) \times [C_a(t_0) - X_a(t_0)]$$

Luego comparando con la ecuación del coeficiente de restitución se llega a una ecuación donde puede despejarse j (1):

$$j = \frac{-(1+\xi) \cdot V_{rel}^{-}}{\frac{1}{M_a} + \frac{1}{M_b} + N(t_0) \cdot \{\left(W_a^{-} + I_a^{-1}(t_0) \cdot [R_a \times N(t_0)]\right) \times R_a + \cdot \left(W_b^{-} + I_b^{-1}(t_0) \cdot [R_b \times N(t_0)]\right) \times R_b\}}$$

Donde $R_a = C_a(t_0) - X_a(t_0)$ y $R_b = C_b(t_0) - X_b(t_0)$. [16]

1.5 Ecuaciones para el Movimiento de un Cuerpo Rígido

Finalmente, después de haber analizado todas las características físicas de un cuerpo rígido se puede definir que el estado de un cuerpo rígido está dado por su posición y su orientación (describiendo la información espacial) y el momento lineal y angular (describiendo la información de la velocidad).

La masa M de un cuerpo y el tensor de inercia espacial de un cuerpo I_{body} son constantes, asumiendo que se conoce cuando va a comenzar la simulación. En cualquier momento dado, las cantidades $I(t), W(t) y V(t)$ serán computadas como: [18]

$V(t) = \frac{P(t)}{M}$, $I(t) = R(t) I_{body} R(t)^T$, y $W(t) = I(t)^{-1} L(t)$

1.6 Obtención de un nuevo estado en la Simulación

La obtención de un nuevo estado en la simulación se puede obtener por un algoritmo que funcione paso a paso. [16] Teniendo el estado de un cuerpo en el tiempo t, la simulación deberá calcular el nuevo estado del cuerpo un tiempo después. De esta forma el estado final del paso anterior será la condición inicial del próximo paso a realizar.

Si k indica el número de pasos en el que se encuentra la simulación, teniendo el estado actual del cuerpo por el vector $Y(t_k)$ y su derivada hay que enfrentar el problema de cómo encontrar $Y(t_{k+1})$. Esto se resuelve por métodos numéricos de resolución de ecuaciones diferenciales, como por ejemplo los métodos de Euler, de Runge-Kutta o de Paso Aditivo, entre otros. [16]

Estos métodos analizan la obtención de una aproximación a una función $Y(t)$ en un intervalo [a,b] calculando su valor para valores de tiempo t_k. Esta función viene dada por la ecuación: [18]

$Y' = f(t, y)$, con $Y(a) = Y_0$

Siendo $Y(t)$ un vector de funciones por lo que se deben resolver cuatro problemas de este tipo donde la función $Y(t)$ en cada uno es $X(t)$, $q(t)$ o $R(t)$, $P(t)$ y $L(t)$ respectivamente.

1.6.1 Método de Euler

El método de Euler es el método más básico de las técnicas de integración numérica. Solamente alcanza un 100 % de exactitud si la tasa de cambio de $Y(t)$ es constante durante todo el tiempo. Pero como esto no ocurre en la mayoría de los casos este método no es lo suficientemente bueno[22].

El método de Euler [16] comienza dividiendo el intervalo [a,b] en M subintervalos del mismo tamaño usando la partición dada por los siguientes puntos.

$t_k = a + kh$ Para $k = 0,1,2,...M$ siendo $h = \frac{b-a}{M}$

Donde h es el tamaño de paso (que en este caso representa tiempo). En la simulación el tamaño del paso puede ser constante o no. Sin embargo, este análisis es útil porque cada paso de la simulación será como el primero en un problema diferencial diferente con un h propio.

Se procede ahora a resolver aproximadamente $Y' = f(t,y)$ en $[t_o, t_M]$ con $Y(t_o) = Y_0$

Suponiendo que $Y(t), Y(t)'$, $Y(t)''$ son continuas y usando el Teorema de Taylor para desarrollar $Y(t)$ alrededor de $t = t_0$, para cada punto t existe un punto c_1 entre t_o y t tal que $Y(t) = y(t_0) + y'(t_0)(t - t_0) + y'' \frac{(c_1)(t - t_0)^2}{2}$.

Al sustituir $Y'(t_0) = f(t_0, y(t_0))$ y $h = t_1 - t_0$ el resultado es:

$$Y(t_1) = y(t_0) + hf(t_0, y(t_0)) + y''(c_1)\frac{h^2}{2}$$

Si el tamaño de paso h es suficientemente pequeño, entonces se puede despreciar el término que contiene h^2 y obtener:

$$Y(t_1) \approx y_1 = y(t_0) + hf(t_0, y(t_0))$$

Que se llama aproximación de Euler. Por despreciar h^2 se dice que hay un error de truncamiento local de orden 2 o O(h^2) y un error acumulativo o global de orden O(h). Cuanto más grande sea el exponente de h a despreciar menos será el error ya que h es cercano a 0. Generando el proceso se obtienen una sucesión de puntos que se aproximan a la gráfica de la solución $Y = Y(t)$.

El paso general del método de Euler es:

$$t_{k-1} = t_k + h \qquad Y_{k+1} = Y_k + hf(t_k, y_k) \qquad \text{Para } k = 0,1,.....M-1.$$

En cada obtención de un valor de Y_{k+1}, se usa el valor anterior Y_k solamente, de forma que los errores en cada paso se van acumulando hasta que el error acumulado se hace demasiado importante. Las consecuencias de este error pueden verse en la simulación cuando aparecen fuerzas grandes que hace que los cuerpos empiecen a tener comportamientos extraños (parece que tiemblan) y a medida que pasa el tiempo empeora. [16]

1.6.2 Método del punto medio

Este método es una simple modificación del método de Euler. El método consiste en usar el método de Euler para predecir un valor de Y en el punto medio del intervalo:[23]

$$y_{i+1/2} = y_i + f(t_k, y_k) \cdot \frac{h}{2}$$

Este valor predicho se usa para calcular una pendiente en el punto medio del intervalo:[23]

$$y_{i+1} = y_i + f(x_{i+1/2}, y_{i+1/2}) \cdot h$$

Este método tiene un error global final de orden O (h^2) [23]

1.6.3 Método de Runge-Kutta o de Paso Aditivo

El método de Runge-Kutta [16] se construye a partir de un desarrollo por Taylor de la función $Y(t)$. Existen diferentes métodos de Runge-Kutta de diferentes órdenes. El método de orden N es que tiene un error global final del mismo orden O (h^N). Esto se logra evaluando la función y en varios puntos, en cada paso. Cuando el orden es de N=4, el método es bastante preciso, estable y fácil de programar, aun más la mayoría de los expertos dicen que no es necesario trabajar con métodos de orden superior porque el aumento de costo computacional no compensa la mayor exactitud.

Consiste en calcular la aproximación Y_{k+1} de la siguiente manera:

$$Y_{k+1} = y_k + w_1 k_1 + w_2 k_2 + w_3 k_3 + w_4 k_4$$

Donde k_1, k_2, k_3, k_4 son de la forma:

$$k_1 = hf(t_k, y_k)$$
$$k_2 = hf(t_k + a_1 h, y_k + b_1 k_1)$$
$$k_3 = hf(t_k + a_2 h, y_k + b_2 k_1 + b_3 k_2)$$
$$k_4 = hf(t_k + a_3 h, y_k + b_4 k_1 + b_5 k_2 + b_5 k_3)$$

Emparejando estos coeficientes con los del método de la serie de Taylor de orden N=4 de manera que el error de truncamiento local sea de orden O(h^5), se obtiene un sistema de ecuaciones que lleva a los valores de la solución que para las variables a_i y w_i Son:

$$a_2 = \frac{1}{2}, a_3 = 1, b_1 = \frac{1}{2}, b_3 = \frac{1}{2}, b_4 = 0, b_5 = 0, b_6 = 1,$$

$$w_2 = \frac{1}{6}, w_2 = \frac{1}{3}, w_3 = \frac{1}{3}, w_4 = \frac{1}{6}$$

Con estos valores se obtiene la fórmula para el método de Runge-Kutta de orden N=4 estándar. A partir del punto inicial (t_0, y_0) se genera la sucesión de aproximación usando la formula recursiva:

$$Y_{k+1} = y_k + \frac{h(f_1 + 2f_2 + 2f_3 + f_4)}{6}$$

Donde:

$$f_1 = f(t_k, y_k)$$

$$f_2 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2} f_1)$$

$$f_3 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2} f_2)$$

$$f_4 = f(t_k + h, y_k + h f_3)$$

Como puede verse al hacer un paso de tamaño h en el tiempo, la función f es evaluada en los tiempo $t_k, t_k + \frac{h}{2}, t_k + h$. Esto requiere un mayor número de cálculos que el método de Euler, pero a menos que se esté realizando una simulación muy simple que no tenga problemas potenciales de instabilidad, la mejora en la reducción de error es considerable y vale la pena realizarla.

Este método tiene la peculiaridad de adecuarse al tipo de función f de forma que el error en la resolución se mantiene bajo y acotado. El método aumenta la estabilidad de la simulación, pero tiene la desventaja de que el cálculo de la estimación de error consume procesamiento.

1.7 Motores Físicos

La idea de delegar el modelo físico utilizado por los juegos o los simuladores en una biblioteca de una tercera parte es algo que se está volviendo muy común. [24] En el mundo muchas computadoras modernas y juegos de video utilizan motores físicos para producir realistas animaciones del movimiento de

los objetos y personajes. [25] Entre los motores físicos más populares podemos encontrar los siguientes: *NovodeX, Newton, Tokamak, ODE, PhysX, Endorphin y Ragdoll.* [24]

De este grupo se presenta una breve documentación de algunos de ellos presentando sus características generales, los juegos que lo utilizan, plataforma y clases principales.

1.7.1 NovodeX Physics SDK

Novodex [24] es una biblioteca muy completa de simulación de entornos físicos en tiempo real. Provee una demostración llamada *Rocket* que es muy completa e ilustrativa así como un SDK provisto de gran cantidad de ejemplos y buena documentación. Recientemente la biblioteca cambió de tipo de licencia permitiendo ser utilizada libremente para fines no comerciales.

Plataforma: Windows, Linux, consolas.

Juegos que utilizan este motor:

- Unreal engine v3 (http://www.unrealtechnology.com)

Características principales:

Asignadores de memoria: Una característica a destacar es la posibilidad de especificar una clase para la asignación de memoria (*allocator*); esta posibilidad es bienvenida particularmente en el desarrollo para consolas debido a que usualmente se suele utilizar un asignador propio que gestiona toda la memoria y no los operadores clásicos *new/delete*.

Depurador visual: Otra característica interesante es la posibilidad de especificar una clase para la depuración visual. En pocas palabras es una clase que especifica métodos para el dibujo de ciertas primitivas que luego utiliza el motor de física para mostrar distintas fuerzas, *bounding boxes*, etc. Esto facilitaría mucho la depuración de un juego.

Clases principales:

Todas las clases de la biblioteca poseen el prefijo Nx, no se hace uso de *namespaces*. La clase principal que representa una escena (mundo, simulador) es **NxScene**. Es posible crear distintas escenas simultáneamente separando dominios de colisión. Una escena no posee límites espaciales, es simplemente una agrupación de actores, uniones y efectores. En una escena la gravedad especificada afecta a todos los objetos de la misma; si se desea crear un juego donde ciertos objetos sean afectados por una gravedad y otros objetos sean afectados por otra, entonces habrá que crear dos escenas distintas. La detección de colisión en una escena puede ser deshabilitada.

Un objeto **NxActor** representa, hoy día, un cuerpo rígido, otras APIs utilizan directamente el nombre de clase RigidBody o similar para referenciarlo, sin embargo Novodex prefiere no hacerlo así ya que el día de mañana pretenden incorporar la simulación de fluidos a su biblioteca que también serían objetos del tipo NxActor. El diagrama muestra las clases principales de la biblioteca Novodex.

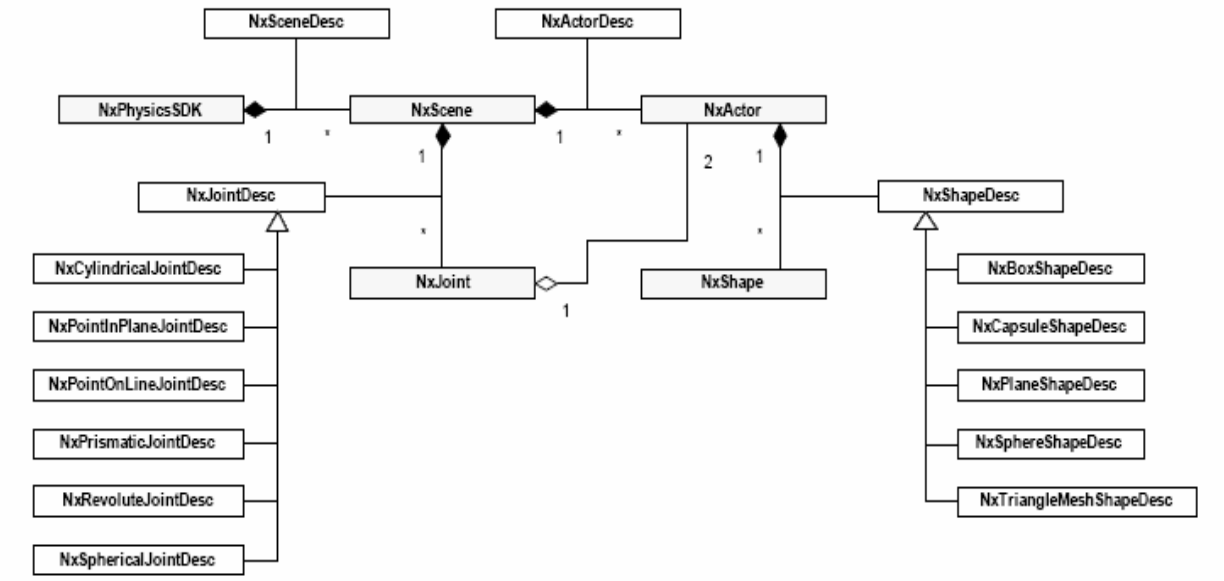


Figura 7 Diagrama de clases de la biblioteca física NovodeX

1.7.2 Newton Game Dynamics

Newton Game Dynamics [24] es una solución integrada para la simulación de entornos físicos en tiempo real. Es libre, pero de código cerrado (*closed source)* para la simulación realista de cuerpos rígidos en juegos y otras aplicaciones en tiempo real. La licencia de *Newton Game Dynamics SDK* permite a los desarrolladores incorporar libremente el motor dentro de su proyecto personal o productos comerciales siempre que sea dado en los créditos y sea distribuido solamente como parte del programa de software compilado, que no es en sí mismo el motor físico. [17]

El API provee gestión de escenas, detección de colisiones, comportamiento dinámico de objetos. Es pequeña, estable y fácil de utilizar. [24]

Plataforma: Windows

1.7.3 Tokamak Game Physics SDK

Tokamak [24] es una biblioteca física de alto rendimiento diseñada específicamente para juegos con una interfaz de programación muy sencilla (a costa de pérdida de flexibilidad). Como todas las bibliotecas, basan sus modelos en cuerpos rígidos que especifican restricciones y soporta detección de colisiones. El asunto es que las primitivas de colisión son bastante pocas y su implementación no facilita la creación de particiones de espacio (para implementación de *quadtrees*, *octrees*, etc.). La interfaz general de la biblioteca está escrita en C++.

Plataforma: Windows

Juegos que utilizan este motor:

- Hollow (http://games.zootfly.com)
- Bontago (http://www.bontago.com/)

Características principales:

Es bastante limitada en funcionalidades, sin embargo esto no impide implementar las funcionalidades más deseadas de una biblioteca física (colisiones, automóviles y *ragdolls*). A diferencia de Novodex, que hace

un uso intensivo de métodos virtuales, la biblioteca Tokamak los evita como decisión de diseño para priorizar el rendimiento. En su lugar, cuando la biblioteca debe realizar llamados al motor o al juego se utilizan funciones callback tipo C.

Clases principales:

La clase principal de la biblioteca es **neSimulator** (el equivalente a NxScene en Novodex), para crear una simulación se debe utilizar un método estático de dicha clase llamado **CreateSimulator**.

Luego los objetos de la simulación pueden ser básicamente dos, los cuerpos rígidos (**neRigidBody** que son los cuerpos que reaccionan y colisionan con el entorno y los cuerpos animados (**neAnimatedBody**) que son los cuerpos que colisionan con el entorno pero que él mismo ni otros objetos lo afectan.

Las uniones se encuentran representadas por la clase **neJoint**. El tipo de unión puede ser especificado luego que la misma es creada (por el método **CreateJoint** de **neSimulator**) por medio del método **SetType**.

La siguiente figura muestra el diagrama de clases de la Biblioteca Tokamak:

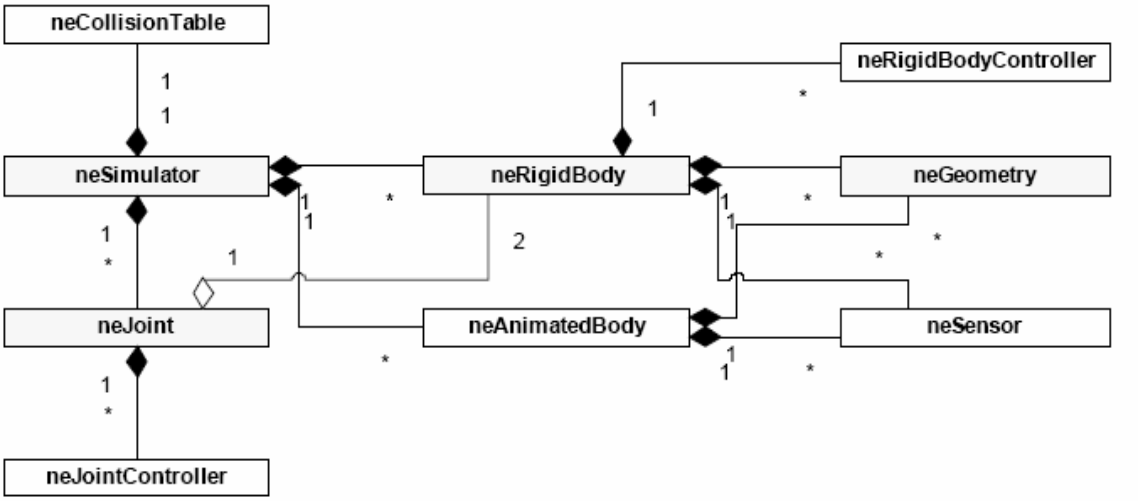


Figura 8 Diagrama de clases de la biblioteca física Tokamak

1.7.4 Open Dynamics Engine

Open Dynamics Engine (ODE) [24] es una biblioteca de código abierto, excelente para la simulación de la dinámica de cuerpos rígidos articulados. Por ejemplo, es buena para la simulación en vehículos, criaturas con articulaciones, y el movimiento de objetos en entornos de Realidad Virtual. Es rápido, flexible, y robusto, incorpora la detección de colisiones. Es además bastante estable.

La biblioteca se encuentra escrita en C++ posee una interfaz C que facilita la integración. Es posible descargar binarios para Windows con precisión simple (uso de floats) y precisión doble (uso de doubles).

Plataforma: Windows, Linux, MAC OS X. Se distribuyen las fuentes de la biblioteca que están escritos en ANSI C++.

Juegos que utilizan este motor:

- Stalker (http://www.stalker-game.com)
- Monster 4x4 (http://www.monster4x4.com/)
- Racer (http://www.racer.nl)
- Exceed (http://sourceforge.net/projects/exceed)
- Gryps Game Engine (http://engine.hlw.co.at/)

Características principales:

La biblioteca agrupa familia de funciones por prefijo, lo cual facilita la búsqueda de funcionalidades comunes a este tipo de bibliotecas. Usualmente cuando se crea algún tipo de entidad en ODE esta retorna un identificador que es un número entero, luego para modificar algunas de sus propiedades se debe utilizar una función y pasarle dicho identificador como primer parámetro.

El grupo de funciones dWorld representan lo que en Novodex es NxScene y lo que en Tokamak es neSimulator. Se pueden poseer cuantos mundos se quieran, sin embargo no es necesario crear distintos mundos para separar dominio de colisiones ya que ODE soporta múltiples espacios por mundos.

La función para avanzar el tiempo en ODE (equivalente a **startRun** de Novodex o **Advance** de Tokamak) se denomina **dWorldStep** o **dWorldQuickStep** en función de la relación velocidad/precisión que deseemos obtener.

El siguiente diagrama no es realmente un diagrama de clases ya que **la interfaz de la biblioteca se encuentra especificada por funciones**. Sin embargo, se optó agrupar las funciones con mismo prefijo que representan un mismo concepto como si fuese una clase con el fin de otorgar una mejor comparativa de las bibliotecas expuestas.

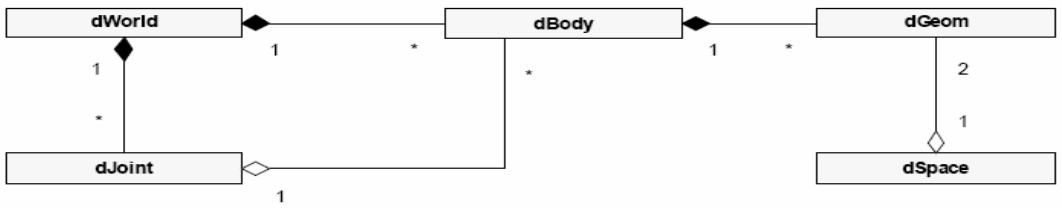


Figura 9 Diagrama de clases de la Biblioteca Física ODE

Clases principales:

La clase dWorld representa la agrupación de funciones "dWorld" por ejemplo: dWorldCreate, dWorldDestroy, dWorldSetGravity, entre otras.

El mundo de objetos esta contenido por cuerpos rígidos y uniones entre ellos. Los objetos que están en diferentes mundos no pueden interactuar, por ejemplo cuerpos rígidos desde dos mundos diferentes no pueden colisionar.

Todos los objetos en el mundo existen en el mismo tiempo, de esta manera una razón para usar mundos separados es simular sistemas a diferentes velocidades. Muchas aplicaciones podrían sólo necesitar un mundo.

La clase dBody representa la agrupación de funciones "dBody" por ejemplo: dBodyCreate, dBodyDestroy, dBodyDisable, dBodyGetData, entre otras. Es aquí donde están representadas varias de las propiedades de los cuerpos rígidos que desde el punto de vista de la simulación pueden ser interesantes: vector de posición, velocidad lineal, orientación, velocidad angular, masa, entre otras.

La clase dJoint representa la agrupación de funciones "dJoints" por ejemplo: dJointCreateBall, dJointCreateHinge, dJointCreateContact, entre otras.

Es aquí donde se agrupan un conjunto de funciones que permiten conectar dos objetos o más y realizar operaciones entre ellos, teniendo cada uno información certera de la posición y orientación relativa del otro u otros.

Conclusiones

En este capítulo se trataron los temas fundamentales a tener en cuenta para simular la dinámica de los cuerpos rígidos en los juegos y simuladores, se mostraron los principales factores que influyen en el desarrollo de este tema brindando aspectos matemáticos y físicos necesarios que sirven para una mejor documentación del proyecto a realizar. Se mostraron las principales técnicas actuales para lograr animaciones físicas lo más realista posible.

Se trabajó también en la investigación de cómo funcionan algunos motores físicos implementados actualmente para Sistemas de Realidad Virtual y específicamente como se asume en ellos la dinámica de los cuerpos rígidos.

Con el estudio realizado del tema se tienen todas las condiciones para pasar a la siguiente fase del proyecto que sería analizar las características del sistema a construir definiendo de forma más específica y clara el objeto de estudio.

2

Capítulo 2 Características del Sistema

Introducción

En el presente capítulo se analizan las características del sistema a realizar, haciendo una propuesta de solución al problema presente, se definen los procesos involucrados con el objeto de estudio, realizando un Modelo del Dominio que agrupa todos los conceptos necesarios para lograr capturar de forma correcta los requisitos del módulo a realizar.

Se enumeran y detallan los requisitos funcionales y no funcionales, que permiten tener una concepción general del módulo, se exponen mediante el Diagrama de Casos de Uso del Sistema los actores y los procesos involucrados en el sistema, definiendo las relaciones e interacción que existe entre ellos.

2.1 Objeto de estudio

Es **objeto de estudio** de este proyecto es la dinámica de los cuerpos rígidos en los simuladores y juegos virtuales y **campo de acción** que abarca este trabajo es la incorporación de la dinámica de los cuerpos rígidos a la herramienta Scene ToolKit desarrollada en la Facultad 5 de la UCI.

2.2 Problema y situación problémica

En la actualidad no existe ningún módulo en la herramienta “Scene Toolkit” existente en la Facultad 5 de la UCI que soporte las leyes físicas ni para el mundo en general, ni particularmente para los cuerpos rígidos. Por esta causa se pierde realismo en los ambientes virtuales que se desarrollan, sin llegar a lograr movimientos reales mediante fuerzas simuladas aplicadas a los objetos.

Por tanto el **problema** a resolver por este tema de tesis es la inexistencia de un modelo físico-matemático que soporte la dinámica de los cuerpos rígidos en los entornos virtuales que se desarrollan con la herramienta “Scene ToolKit” en los proyectos de la Facultad 5 de la UCI

2.3 Objeto de automatización

Como se mencionó anteriormente el módulo a crear va a formar parte de la herramienta para SRV “Scene ToolKit”, de esta herramienta se tienen identificados los conceptos que van a estar vinculado al módulo. El manejo de los objetos de la escena se realiza a través de un grafo de escena. Este grafo esta formado por un grupo de nodos, dentro de los cuales están los de contenido (“nodos geometría”, “nodos cámara”, “nodos luz”, “nodos personaje” y “nodos hueso”) que contienen toda la información necesaria del objeto que contienen. Cada nodo tiene el estado geométrico local y global en el que se encuentra (posición, orientación, y escalado).Todos estos nodos según sus características tendrán asociado un controlador que básicamente lo que haría es actualizar el estado geométrico local del nodo que contiene en un instante dado de la simulación. Deben existir distintos tipos de controladores, controlador de esqueleto, controlador de camino para objetos con trayectorias predefinidas en la escena, y **controladores físicos** que serían los que actualizan el estado geométrico de un objeto según las leyes físicas que lo rigen, siendo este el que va a formar parte del módulo a realizar.

2.4 Información que se maneja

Los cuerpos que se tendrán en cuenta para asignarle un **controlador físico** son los llamados cuerpos rígidos, estos son los cuerpos que no permiten modificaciones en su estructura, es decir, que no se deforman. Por tanto no se tendrán en cuenta deformaciones de un cuerpo producidas en algún choque o de fuertes fuerzas de impacto [4].

Así queda definido que de los cuerpos rígidos se almacenarán los siguientes datos definiendo su **estado físico**:

- Vector de Estado
- Cantidades Auxiliares
- Constantes del Cuerpo

2.4.1 Vector de Estado

El vector de estado debe contener en cualquier momento la información necesaria sobre el estado de cuerpo de traslación y rotación. Se define para los cuerpos rígidos un *vector de estado* $Y(t)$ con la siguiente información:

$$Y(t) = \begin{pmatrix} X(t) \\ q(t) \\ P(t) \\ L(t) \end{pmatrix}$$

Figura 10 Vector de estado

Así, el estado de un cuerpo rígido esta dado por su posición $X(t)$ y su orientación $q(t)$ (describiendo la información espacial) y el momento lineal $P(t)$ y angular $L(t)$ (describiendo la información de la velocidad), [18]

- $X(t)$ es el vector de **posición** (x, y, z) del cuerpo en coordenadas globales como un punto de referencia. Puede corresponderse con el centro de masa del cuerpo.
- $q(t)$ es la **orientación** del cuerpo, representada por un *quaternion* de rotación.
- $P(t)$ es el vector de **cantidad de movimiento lineal**.
- $L(t)$ es el vector de **cantidad de movimiento angular**.

2.4.2 Constantes del Cuerpo

Estas cantidades son las que pueden ser definidas al principio de la simulación debido a que no tienen variación en ningún instante posterior se pueden mencionar como las más importantes las siguientes:

- **Masa** del cuerpo.
- I_{body} es la **Matriz de Inercia**. Esto es una matriz de 3x3 que describe como la masa del cuerpo es distribuida alrededor de su centro de masa.

La masa es la inercia del cuerpo con respecto a las fuerzas que se apliquen en su centro de masa, análogamente la matriz de inercia es la inercia con respecto a las torques aplicadas. Es valido aclarar que el tensor de inercia no es constante, lo que es constante es la matriz de inercia inicial que se usa para calcular el tensor de inercia en un instante dado, por eso también se debe guardar como una magnitud variable el **tensor de inercia**.

2.4.3 Cantidades auxiliares

Las cantidades auxiliares que se almacenan son las que guardan la relación que existe entre el vector de estado del cuerpo y las fuerzas y torques que se aplican.

Si se deriva el vector de estado $Y(t)$ se obtiene $\frac{d}{dt}Y(t)$:

$$Y'(t) = \begin{pmatrix} V(t) \\ \frac{1}{2}W(t)q(t) \\ F(t) \\ T(t) \end{pmatrix}$$

Figura 11 Derivada del vector de estado

De esta forma, la derivada de la posición del centro de masa es su velocidad, la de la rotación se obtiene de la velocidad angular y el *quaternion* de la rotación actual, la del momento lineal de las fuerzas aplicadas sobre el centro de masa con la fuerza y la del momento angular de los torques aplicados. [16]

- $V(t)$ es el vector que almacena la **velocidad lineal** del cuerpo.
- $W(t)$ es la **velocidad angular** del cuerpo.
- $\frac{1}{2}W(t)q(t)$ define la relación que se establece entre el *quaternion* de rotación y la velocidad angular que describe como cambia la orientación del cuerpo en el tiempo, a esta relación le llamaremos para identificarla **spin**.
- $F(t)$ es el vector que almacena las **fuerzas** aplicadas sobre el cuerpo.
- $T(t)$ es el vector que almacena el **torque** producido por las fuerzas que actúan sobre el cuerpo.
- $I(t)$ es una matriz 3x3 que guarda el **tensor de inercia** del cuerpo.

En cualquier momento dado se pueden calcular $I(t), W(t) y V(t)$ por la siguiente fórmula:

$$V(t) = \frac{P(t)}{M} \text{ , } I(t) = R(t)I_{body}R(t)^T \text{ , y } W(t) = I(t)^{-1}L(t)$$

2.4.4 Propiedades del mundo físico

Además de controlar información de los cuerpos rígidos es necesario tener en cuenta las propiedades y condiciones que deben existir en el entorno para que estos cuerpos mantengan su comportamiento según el mundo físico, es por ellos que se deben almacenar algunas propiedades generales que afectarán a todos los cuerpos de la escena, ellas son las siguientes:

- Gravedad: constante de la aceleración de la gravedad que se desee utilizar.
- Superficies: lista de superficies físicas que se tendrán en cuenta para determinar las fuerzas actuantes sobre el cuerpo cuando este apoyado sobre la misma. Se debe almacenar de cada superficie sus límites dados por un punto mínimo y un punto máximo, y los coeficientes de fricción estático y cinético.
- Medios o ambientes: lista de ambientes que se tendrán en cuenta para ejercer una fuerza resistiva sobre el objeto que se encuentre en alguno de ellos. De cada ambiente se almacena su tipo, y un punto mínimo y un punto máximo que delimitan el volumen que los contiene.

2.5 Propuesta de sistema

Los motores físicos analizados en el capítulo 1 de una forma u otra soportan la dinámica de los cuerpos rígidos, entonces valdría preguntarse porque en lugar de diseñar un módulo propio desde cero no se hace a partir de la utilización de uno de estos ya realizados.

Las razones son las siguientes:

Novedex es de código cerrado y su utilización es autorizada a fines no comerciales solamente, lo que limitaría su utilización para desarrollar simuladores y juegos para comerciar, que es el objetivo fundamental de los proyectos de la Facultad.

Newton es también de código cerrado aunque su uso es libre.

En ambas bibliotecas la característica de ser de código cerrado no permitiría incorporar nuevas funcionalidades que sean de interés a los distintos proyectos que utilicen la herramienta por lo tanto limitaría su uso a desarrollador simulaciones específicas para un proyecto determinado.

En el caso de *Tokamak* está realizada solamente para juegos, donde el nivel de realismo que al que se desea llegar no es tan grande como en el caso de simulaciones de conducción o de procesos físicos, por lo que se limitaría su uso a determinados productos.

En el caso de *ODE* es libre, se tiene acceso al código, pero su uso se limita a escenas donde se controlen sólo los cuerpos rígidos, el control del tiempo lo hace internamente, y actualiza todos los cuerpos rígidos en un ciclo de escena propio, y la herramienta en desarrollo debe prever el control de otros cuerpos, como los cuerpos deformables, los personajes, y otros que por sus características tengan trayectoria y comportamientos fijos y propios en la escena, la herramienta también implementa en sí el control del tiempo y es la que hace directamente la actualización del estado de los objetos.

Por todas estas razones surge la idea de desarrollar un módulo desde cero que incorpore la dinámica de los cuerpos rígidos, y que en un futuro se le puedan añadir nuevas funcionalidades.

La propuesta que se brinda es la siguiente:

El módulo a desarrollar debe tener en cuenta dos partes fundamentales, debe primeramente actualizar el estado de los cuerpos a través del tiempo y como segundo aspecto atender el proceso de respuesta a la colisión que se produce entre ellos.

Es válido recordar que, como se detalló en Capítulo 1 las colisiones que ocurren en la simulación tienen dos fases, la detección de la colisión y la respuesta a ella, existe otro módulo que se encarga de la primera fase, por tanto, ante la información que brinda este módulo de los cuerpos que colisionaron en un instante de tiempo se debe responder a la colisión.

Para actualizar el estado de un cuerpo se debe tener en cuenta las siguientes entradas:

- El estado anterior en el que se encontraba, que sería el vector de estado definido anteriormente.
- Las fuerzas que se apliquen sobre el cuerpo que producirán una nueva variación en el estado.
- El espacio de tiempo desde el paso anterior hasta este nuevo estado.

Después de cada uno de estos pasos se debe procesar si existen colisiones entre los objetos, y este ciclo repetirlo mientras dure la simulación. Para cada nueva iteración el estado anterior es el estado nuevo de la iteración anterior. Así se verá como funciona en la evolución del tiempo la dinámica de los cuerpos cuando se le aplica fuerzas sobre ellos. Esto se aprecia en la figura que se muestra a continuación:[16]

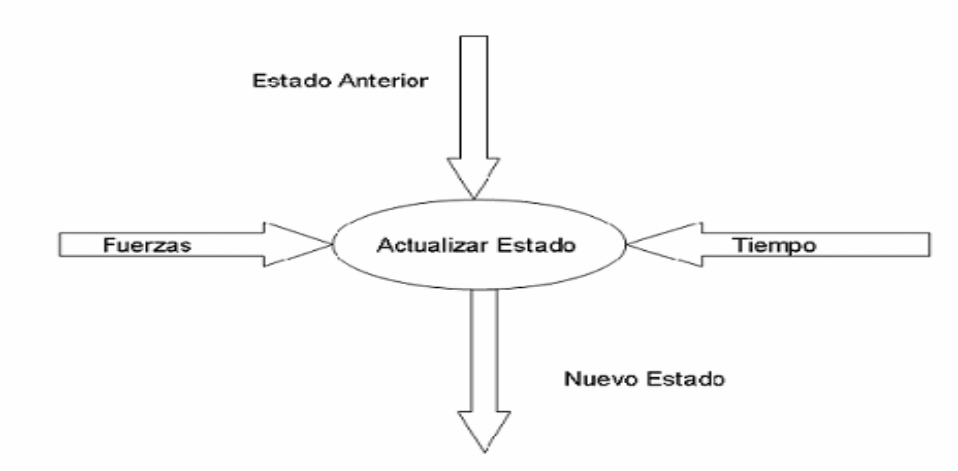


Figura 12 Actualización de un Nuevo estado en la Simulación

El proceso "Actualizar Estado" a partir del vector de estado $Y(t)$ y su derivada $Y'(t)$ se obtendrá un nuevo estado y se recalcularan las cantidades auxiliares definidas anteriormente, luego de un intervalo de tiempo. El método que se utilizará por defecto para resolver estas ecuaciones diferenciales es el de Runge-Kutta que es más rápido y permite realizar más pasos por segundo. Pero se dejara la opción al usuario final de escoger entre los otros dos métodos estudiados (Euler y Punto Medio) para realizar la simulación, esto se tuvo en cuenta, dado que si el entorno que se va a simular es sencillo y no requiere de exactitud en los cálculos de posición y orientación de los cuerpos otro método podría resultar conveniente.

Atender el proceso de respuesta a las colisiones es el paso que sigue después de realizar la actualización del estado de los cuerpos. Lo primero que se analiza es que objetos están colisionando, información que se dará de otro módulo, teniendo en cuenta la información de qué cuerpos están colisionando se procede a realizar una acción que intentará separarlos.

El método del impulso es el que se llevará a cabo y consiste en aplicar una fuerza producto del choque a los cuerpos en colisión que tiende a separarlos. A esta fuerza que es muy grande y se aplica por un período de tiempo muy chico, se le asocia un impulso, de forma que a los cuerpos les causa un cambio de velocidad instantánea. Este impulso se calcula en función de la velocidad, de las masas de los cuerpos y del coeficiente de restitución como se analizó en el epígrafe 1.4.1.

Ahora se muestra el algoritmo de simulación final en un seudo-código:

Inicio

1. Inicializar propiedades del mundo físico.
2. Inicializar estados de los cuerpos.
3. Para cada cuerpo con controlador físico asociado que lo identifique:
   3.1 Calcular tamaño de paso Δt.
   3.2 Determinar la sumatoria de las fuerzas aplicadas.
   3.3 Aplicar fuerza resultante al cuerpo.
   3.4 Calcular la derivada del vector Y(t), y Y(t) con la fuerza total.
   3.5 Obtener Y(t+Δt).
   3.6 Si un cuerpo colisionó con otro.
   3.7 Calcular la Vrel entre los cuerpos que colisionaron.
   3.8 Si Vrel <=0 aplicar impulso.

   Repetir el paso 3 durante toda la simulación

Fin

Este seudo-código contiene un ciclo principal donde, para cada uno de los cuerpos que es de interés que estén sometidos a leyes físicas, se le calcula en cada paso de la simulación la suma de fuerzas que actúan sobre él y se les separa entre las que actúan sobre el centro de masa y las que producen torque (rotación del cuerpo), ambos juntos con el vector de estado se utilizan como punto de partida para calcular el nuevo estado de la simulación un instante posterior. En el ciclo, si ocurre alguna colisión se le da respuesta a la misma mediante el método del impulso.

2.6 Reglas del Negocio

Los volúmenes de colisión que se tendrán en cuenta para los cuerpos son: caja, cilindro y esfera, es por eso que el cálculo del tensor de inercia sólo se realizará para estos volúmenes.

El estado geométrico inicial del nodo (posición y orientación inicial) será también los valores del vector de estado asociado al controlador físico de este nodo, en ningún caso se le pondrá una posición y orientación inicial distinta.

Sólo se tendrán en cuenta cómo fuerzas incidentes en el cuerpo: fuerza de gravedad, fuerza de rozamiento, fuerza normal, fuerza de usuario, y en los casos que se indique resistencia del aire, cualquier otra fuerza que pueda incidir sobre el cuerpo en la realidad aquí no será simulada automáticamente.

Para el cálculo de la fuerza de rozamiento actuando sobre un cuerpo sólo se tendrá en cuenta el coeficiente de fricción de la superficie donde se encuentra, no el coeficiente de fricción del cuerpo en sí mismo, que en el mundo real también existe.

Se considerará que las fuerzas que actúen sobre el cuerpo lo harán en un punto que coincide con su centro de masa, excepto en los casos de las fuerzas que producen torque.

El cuerpo se considerará como una masa puntual, el punto de referencia del cuerpo coincidirá en todos los casos con el centro de masa y el centro de gravedad del mismo. Este punto en coordenadas locales siempre estará en la posición (0, 0, 0).

2.7 Modelo del dominio

Teniendo en cuenta que el módulo a desarrollar forma parte de una herramienta más general y que la incorporación de este módulo a la herramienta no constituye un negocio en sí con todas sus prestaciones y funcionalidades se desarrollará un modelo de dominio, que permitirá mostrar al usuario los principales conceptos que se manejan en el sistema en desarrollo. La realización de este modelo del dominio ayuda a todas las personas involucradas en el proyecto (usuarios o desarrolladores) a entender vocablos y términos en común que se utilizaran para la realización del módulo, a realizar de forma correcta la captura de

requisitos, y a agrupar todos los conceptos necesarios del objeto de estudio que permiten identificar algunas clases que formarán parte del diseño final del módulo.

Primeramente se dará un glosario de términos de todos los conceptos que se manejan en el modelo del dominio facilitando la familiarización con los términos utilizados.

Se denominará: **Rigid Body** a los objetos de la escena que no tienen deformación en su estructura y que tienen un conjunto de propiedades físicas por la que se regirá su movimiento como pueden ser masa, posición, velocidad lineal y angular, cantidad de movimiento lineal y angular.

Se denominará **Vector State** a la representación de atributos que definen la información espacial del cuerpo.

Se denominará **DVector State** a la representación de atributos que definen la derivada del **Vector State** y que son de gran importancia para actualizar el estado del **Rigid Body.**

Se denominará: **Animation Body** a los objetos de la escena que no se rigen por sus características físicas para su movimiento o que están en algún lugar fijo de la escena, pero que estos cuerpos sí pueden interactuar en algún momento con los cuerpos rígidos.

Se denominará: **Collision Response** al manejo de la respuesta a las colisiones entre un cuerpo rígido (**Rigid Body**) con algún otro cuerpo rígido o con alguno del tipo **Animation Body**.

Se denominará: **Simulation Physic** al manejo de las características físicas generales del mundo como las fuerzas que están actuando o no en cualquier momento de la simulación (gravedad, fricción, resistencia al medio).

Se denominará: **Physic Controller** al controlador que se le atacha a cada cuerpo rígido (**Rigid Body)** de la escena y que debe actualizar cada uno de los estados que toma el cuerpo en el transcurso de la simulación.

El modelo del dominio se describe en el lenguaje de modelado UML, específicamente con un diagrama de clases conceptuales significativas en el dominio del problema.

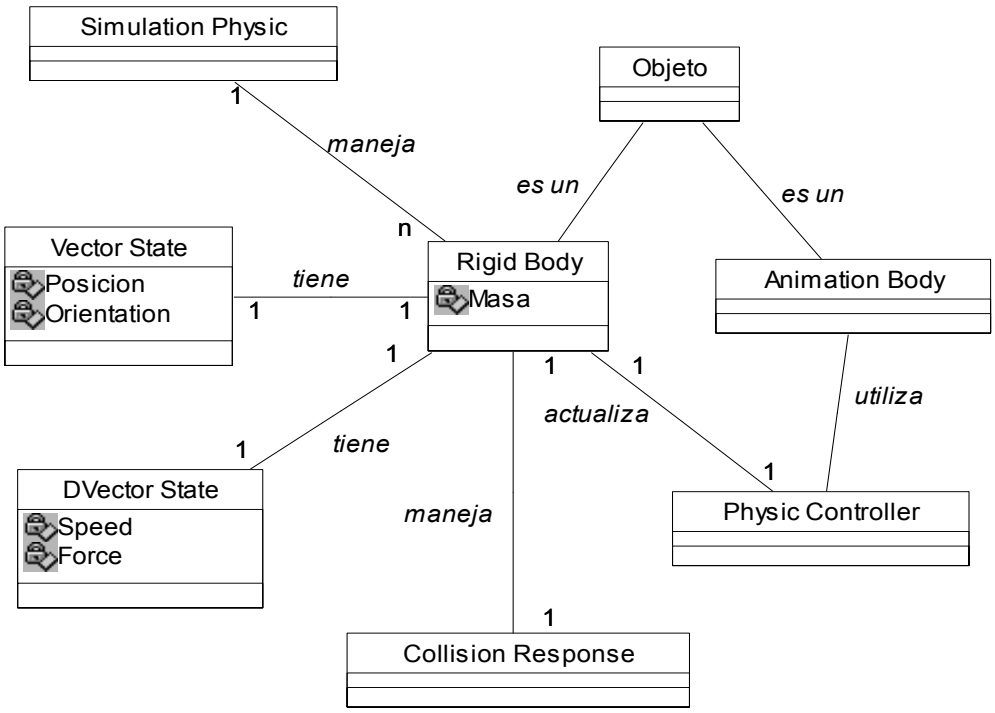


Figura 13 Modelo de Dominio

2.8 Especificación de los requisitos de software

2.8.1 Requisitos Funcionales

Una vez conocidos los conceptos que rodean al objeto de estudio, podemos empezar a analizar ¿Qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo?, para ello enumeraremos a través de requerimientos funcionales las funciones que el sistema deberá ser capaz de realizar. Dentro de ellos se incluyen las acciones que podrán ser ejecutadas por el usuario, las acciones ocultas que debe realizar el sistema, y las condiciones extremas a determinar por el sistema. De acuerdo con los objetivos planteados el sistema debe ser capaz de:

1. Crear mundo físico.
2. Asignar gravedad al mundo creado.
3. Seleccionar el método de integración a utilizar.
4. Asignar el “paso” de la simulación.

   4.1 Asignar el “paso” manualmente (valor fijo).

4.2 Asignar el “paso” del ciclo de escena.

5. Seleccionar los tipos de ambiente en el que se va a trabajar (aire, agua, etc) y el volumen que los contiene a cada uno de ellos.
6. Crear controlador físico.
7. Atachar controlador físico a los CR de la escena.
8. Crear estado físico inicial del CR.
9. Asociar al controlador creado referencia al mundo que pertenece.
10. Determinar las constantes físicas del cuerpo.
    - 10.1 Asignar valor de la masa del cuerpo.
    - 10.2 Asignar dimensiones del volumen de colisión.
    - 10.3 Calcular los momentos principales de inercia inicial.
    - 10.4 Asignar los momentos principales de inercia al tensor de inercia inicial.
    - 10.5 Calcular inversa del tensor de inercia inicial.
11. Determinar las condiciones iniciales del estado.
    - 11.1 Asignar posición inicial.
    - 11.2 Asignar orientación inicial.
    - 11.3 Asignar velocidad lineal inicial.
    - 11.4 Asignar velocidad angular inicial.
    - 11.5 Asignar aceleración inicial lineal.
    - 11.6 Asignar aceleración inicial angular.
    - 11.7 Asignar momento angular inicial.
    - 11.8 Asignar momento lineal inicial.
12. Calcular fuerzas incidentes en el cuerpo.
    - 12.1 Calcular fuerza de fricción.
    - 12.2 Calcular fuerza de gravedad.
    - 12.3 Calcular fuerza que llegue de algún sensor (teclado, mouse).
    - 12.4 Calcular fuerza de resistencia del medio.
13. Calcular la fuerza resultante incidiendo sobre el cuerpo.
14. Calcular fuerzas de reacción.
15. Calcular torque producido por la fuerza resultante.

16. Actualizar el estado del cuerpo.
    16.1 Resolver ecuación de la posición.
    16.2 Resolver ecuación de la orientación.
    16.3 Resolver ecuación de la cantidad de movimiento lineal.
    16.4 Resolver ecuación de la cantidad de movimiento angular.
17. Calcular la velocidad lineal del cuerpo.
18. Calcular Tensor de Inercia del cuerpo.
19. Calcular la aceleración angular del cuerpo.
20. Responder a la colisión entre cuerpos.
    20.1 Asignar valor del coeficiente de restitución de la colisión.
    20.2 Asignar el vector Normal a la colisión.
    20.3 Obtener los datos de los cuerpos que colisionan.
    20.4 Calcular velocidad relativa antes de la colisión.
    20.5 Calcular la magnitud del impulso producto de la colisión.
    20.6 Calcular las nuevas velocidades de los cuerpos que colisionaron.
    20.7 Calcular el nuevo momento lineal con el impulso.
    20.8 Calcular el nuevo momento angular con el impulso.
21. Crear superficie física.
    21.1 Asignar valor del coeficiente de fricción estático y cinético.
    21.2 Asignar valor del vector normal a la superficie.
    21.3 Asignar el valor del punto mínimo de la superficie.
    21.4 Asignar el valor del punto máximo de la superficie.
    21.5 Calcular si un cuerpo esta apoyado o no en la superficie.
22. Adicionar Fuerza.
    22.1 Adicionar una fuerza al cuerpo que actúe de forma permanente sobre su centro de masa.
    22.2 Adicionar una fuerza al cuerpo que actúe sobre un punto distinto de su centro de masa.
    22.3 Adicionar una fuerza al cuerpo que actúe en un instante de tiempo sobre su centro de masa.
    22.4 Adicionar una fuerza al cuerpo que actúe en un instante de tiempo sobre un punto.
    22.5 Adicionar un torque al cuerpo que actúe de forma permanente.
    22.6 Adicionar un torque al cuerpo que actúe en un instante de tiempo.

2.8.2 Requisitos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

Apariencia o interfaz externa.

- **Usabilidad:** Los usuarios que trabajen con el módulo deben tener conocimientos básicos de física y de toda la terminología a fin. El sistema debe estar concebido para que todos los cuerpos rígidos puedan ser controlados por un controlador físico definido por los usuarios de la herramienta. Deberá ser lo suficientemente flexible como para que los futuros usuarios puedan, a partir de los modelos generales implementados, adaptarlo a su problema en particular.
- **Rendimiento:** La modelación físico-matemático de los cuerpos rígidos se va a desarrollar en tiempo real por lo que debe tener alto grado de velocidad ó cálculo de procesamiento de las leyes físico-matemático aplicables al objeto, un rápido tiempo de respuesta, de recuperación, y disponibilidad.
- **Soporte:** En una versión inicial deberá ser compatible con la plataforma Windows, pero debe estar preparado para que con rápidas modificaciones pueda migrar para Linux.
- **Portabilidad:** La implementación de la biblioteca se realizará en C++ con el objetivo de que sea multiplataforma.
- **Legales:** Se regirá por las normas ISO 690.
- **Software:** Se utilizará el IDE Microsoft Visual Studio .NET 2003 sobre el sistema operativo Windows.
- **Diseño e implementación:** Se utilizará el lenguaje de implementación C++. Se regirá por la filosofía de Programación Orientada a Objetos.

2.9 Modelo de casos de uso del sistema.

Utilizando las facilidades que nos brinda el UML, se pueden capturar los requisitos funcionales del sistema y representarlos mediante un diagrama de casos de uso. Para ello tenemos que definir de acuerdo a lo planteado en los epígrafes anteriores, cuales serían los actores que van a interactuar con el sistema, y los casos de uso que van a representar las funcionalidades.

Actor del sistema

Un actor no es parte del sistema, es un rol de un usuario, que puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que interactúa con el sistema. En este caso con el módulo interactúa 1 sólo actor que se define a continuación:

Actores	Justificación
Programador	Este módulo es un subsistema de una herramienta destinada a ser utilizada por programadores para hacer aplicaciones finales, por tanto el programador es el que se beneficiará con las funcionalidades que brinda el módulo, que serían de forma general: asignarle propiedades físicas al mundo e inicializar el estado de los cuerpos, actualizar el estado de los cuerpos, y dar respuesta a las colisiones.

Tabla 1 Actores del Sistema

2.9.1 Diagrama de casos de uso

Teniendo en cuenta los principales procesos que se deben manejar en el módulo, y las dependencias que pueden existir entre ellos, el diagrama de casos de uso del sistema queda organizado de la siguiente forma:

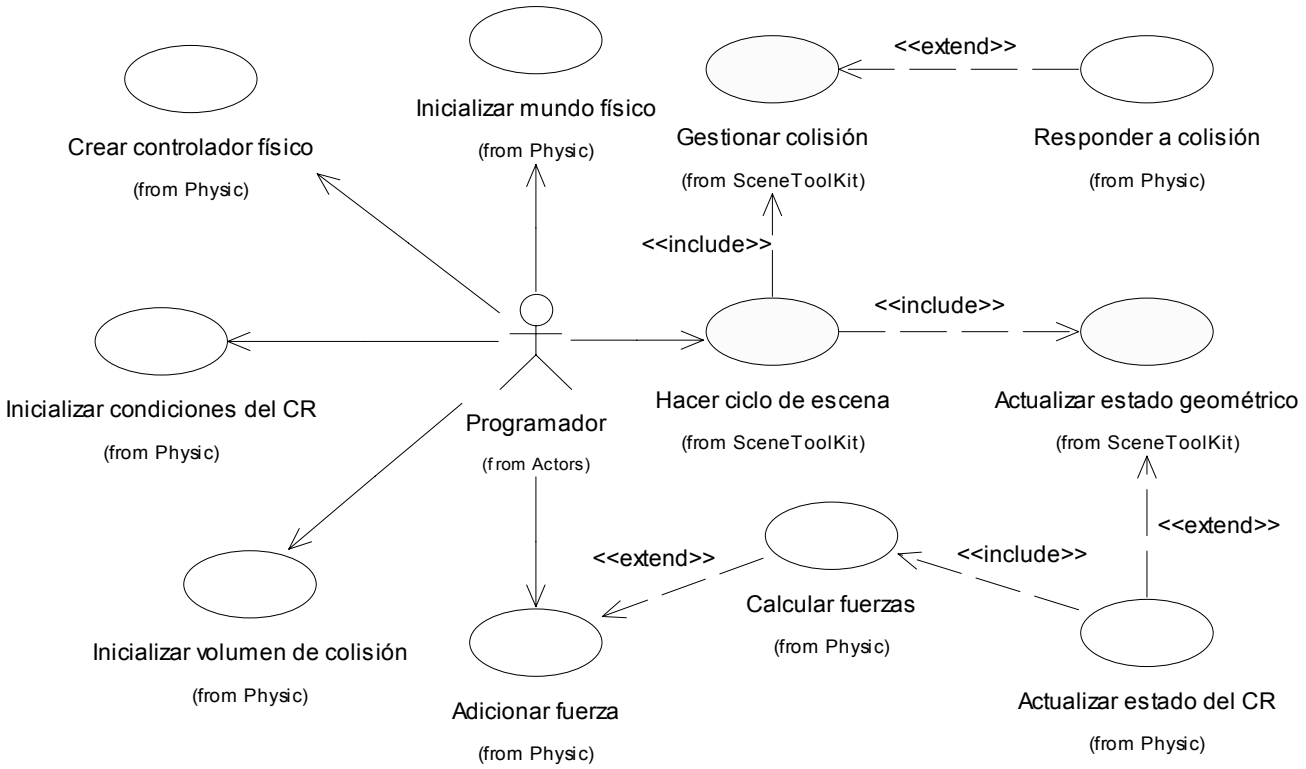


Figura 14 Diagrama de Casos de Uso del Sistema

Los casos de uso “Hacer ciclo de escena”, “Actualizar estado geométrico”, y “Gestionar colisión” no forman parte del módulo, son CU de la herramienta “SceneToolKit”, que de alguna forma utilizan los casos de uso del módulo.

El caso de uso “Hacer ciclo de escena” es el que repite durante toda la simulación actualizando los estados geométricos de los objetos y visualizándolos en el proceso de *render.*

El caso de uso “Gestionar colisión”, es una inclusión del caso de uso “Hacer ciclo de escena”. En este caso de uso se procede a verificar si se ha producido alguna colisión entre los objetos de la escena, si esto sucede se invoca al caso de uso “Responder a colisión” que separa los cuerpos que colisionaron.

El caso de uso “Actualizar estado geométrico”, es una inclusión del caso de uso “Hacer ciclo de escena”, este caso de uso actualiza el estado geométrico del objetos (posición y orientación) según su controlador asociado, si el objeto tiene un controlador físico se invoca al caso de uso “Actualizar estado del CR”, que actualiza el estado del cuerpo teniendo en cuenta las fuerzas que inciden sobre él.

El módulo a crear se va a desarrollar en un único ciclo de desarrollo donde se logrará la realización en los diferentes flujos de trabajo de todos los casos de uso propuestos. Se hará uso de un solo paquete que agrupará toda la información del módulo.

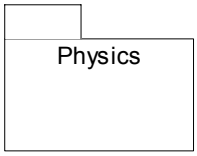


Figura 15 Paquete para la dinámica de los CR

2.9.2 Casos de uso expandidos.

Mediante los casos de uso expandidos se describe paso a paso la secuencia de eventos que los actores utilizan para completar un proceso a través del sistema.

Tabla 2 Caso de Uso expandido: Inicializar mundo físico

Caso de uso		
CU-1	Inicializar mundo físico	
Propósito	Asignarle propiedades físicas al medio que integra el mundo virtual	
Actores	Programador(Inicia)	
Resumen: El caso de uso se inicia cuando el programador solicita crear el mundo físico. Se crea el mundo físico y se le asigna el valor de la gravedad, los tipos de medio en que se trabajará y las diferentes superficies físicas que se van a tener en cuenta para el cálculo de la fuerza de rozamiento. A continuación se especifica el método de resolución de ecuaciones que se va a utilizar y el tamaño del paso en el caso de que este sea fijo. El caso de uso finaliza cuando estas propiedades son inicializadas.		
Referencias	1, 2, 3, 4, 5, 21.1 al 21.4	
Acción del actor		Respuesta del sistema
1- Solicita crear el mundo físico.		2- Se crea el mundo físico.
3- Solicita la inserción del valor de la gravedad al mundo.		4- Se le asigna el valor de la gravedad al mundo.
5- Solicita la inserción de los tipos de medios(aire, agua, etc) que se tendrán en cuenta, para cada medio se introducen los siguientes datos: • Tipo de ambiente. • Punto mínimo del volumen. • Punto máximo del volumen. Nota: Este paso se repite tantas veces como ambientes quiera inicializar el programador.		6- Se crea el medio.
		7- Se le asigna el valor del coeficiente del medio según valores constantes predefinidos.

8- Solicita especificar el método de resolución de ecuaciones que se va a utilizar. (Método de Euler, Método de Punto Medio ó Método de Rungue Kutta.)	9- Se asigna el método a utilizar.
10- Se solicita especificar si el tamaño del paso es fijo o variable.	11- Se asigna este valor. Si se va a utilizar un tamaño de paso fijo ir a “Sección 1”.
12- Se solicita especificar las superficies que van a existir y que se tendrán en cuenta en el mundo físico para producir fuerza de rozamiento, para esto se introducen los siguientes datos de cada superficie: • Coeficiente de fricción estático. • Coeficiente de fricción cinético. • Vector normal al plano correspondiente a la superficie. • Punto mínimo del plano. • Punto máximo del plano. Nota: Este paso se repite cuantas veces como superficies físicas quiera inicializar el programador.	13- Se crea la superficie y se le asignan los valores correspondientes. Y con el punto mínimo y la normal del plano se calcula el valor de la constante D de la ecuación del plano: Ax+By+Cz+D=0.
Sección 1: Tamaño de paso fijo.	
1- Se solicita introducir el tamaño del paso a utilizar.	2- Se asigna el tamaño del paso.
Fin de Sección.	
Precondiciones	-
Poscondiciones	Mundo físico creado con todas sus propiedades.

Tabla 3 Caso de Uso expandido: Crear controlador físico

Caso de uso		
CU-2	Crear controlador físico	
Propósito	Crear un controlador físico y atachárselo al nodo correspondiente.	
Actores	Programador(inicia)	
Resumen: El caso de uso se inicia cuando el programador solicita la creación del controlador físico que es el encargado de actualizar el estado de los cuerpos que se consideren CR en la escena, una vez creado este controlador se le atacha al nodo correspondiente y se le especifica el mundo al que pertenece.		
Referencias	6, 7, 8, 9	
Acción del actor		Respuesta del sistema
1- Solicita crear el controlador físico.		2- Se crea el controlador físico.
		3- Se crea el estado físico del CR asociado al controlador.
4- Solicita atachar el controlador físico creado al nodo correspondiente.		5- Se atacha el controlador físico al nodo correspondiente.
6- Solicita especificar el mundo al que pertenece.		7- Se le asigna una referencia al mundo al que pertenece.
Precondiciones	Tiene que existir al menos un nodo creado para atachar el controlador.	
	Tiene que existir un mundo físico al que pueda pertenecer este controlador.	
Poscondiciones	Queda atachado el controlador físico al nodo correspondiente y asignado la referencia al mundo físico al que pertenezca.	

Tabla 4 Caso de uso expandido: Inicializar condiciones del CR

<table>
<tr><th colspan="4">Caso de uso</th></tr>
<tr><td>CU-3</td><td colspan="3">Inicializar condiciones del CR.</td></tr>
<tr><td>Propósito</td><td colspan="3">Inicializar la posición y orientación del cuerpo y todas sus propiedades físicas.</td></tr>
<tr><td>Actores</td><td colspan="3">Programador (inicia)</td></tr>
<tr><td colspan="4">Resumen: El caso de uso se inicia cuando el programador solicita inicializar la posición, orientación, velocidad lineal y angular inicial del cuerpo y su masa. Después se inicializan las demás propiedades físicas: momento lineal, angular y spin. El caso de uso finaliza cuando se le asignan los valores iniciales al las propiedades físicas de los cuerpos rígidos.</td></tr>
<tr><td>Referencias</td><td colspan="3">11.1 al 11.8</td></tr>
<tr><td colspan="2">Acción del actor</td><td colspan="2">Respuesta del sistema</td></tr>
<tr><td colspan="2">1-Solicita inicializar todas las propiedades físicas del CR, se introducen los siguientes datos:

• Posición.
• Orientación
• Velocidad lineal
• Velocidad angular
• Masa</td><td colspan="2">2- Se le asigna el valor inicial de la posición, orientación, velocidad lineal y angular, y la masa del CR.</td></tr>
<tr><td colspan="2"></td><td colspan="2">3- Se calcula el spin(derivada de la orientación) con la orientación y la velocidad angular inicial.</td></tr>
<tr><td colspan="2"></td><td colspan="2">4- Se calcula el momento lineal inicial.</td></tr>
<tr><td colspan="2"></td><td colspan="2">5- Se calcula el momento angular inicial.</td></tr>
<tr><td>Precondiciones</td><td colspan="3">Tiene que existir al menos un controlador físico para inicializar su estado.</td></tr>
<tr><td>Poscondiciones</td><td colspan="3">Quedan asignados los valores iniciales del estado físico del cuerpo.</td></tr>
</table>

Tabla 5 Caso de Uso expandido: Inicializar volumen de colisión

Caso de uso		
CU-4	Inicializar volumen de colisión	
Propósito	Inicializar volumen de colisión para el cálculo de la matriz de inercia inicial.	
Actores		
Resumen: El caso de uso se inicia cuando el programador solicita crear el volumen de colisión asociado al cuerpo introduciendo las dimensiones del mismo, con estos datos y la masa del cuerpo se calcula la matriz de inercia inicial del cuerpo. El caso de uso finaliza cuando queda calculada la matriz de inercia inicial.		
Referencias	10.1 al 10.5	
Acción del actor		Respuesta del sistema
1- Solicita crear el volumen de colisión asociando al cuerpo introduciendo las dimensiones del mismo.		2- Se crea el volumen de colisión
		3- Se le asignan las dimensiones del volumen de colisión.
		4- Se calculan los momentos principales de inercia inicial.
		5- Se le asignan los momentos principales de inercia al tensor de inercia inicial.
		6- Se calcula la inversa del tensor de inercia inicial.
7- Solicita referenciar al cuerpo rígido su volumen de colisión.		8- Se le asigna una referencia al cuerpo del volumen de colisión asociado a él.
Precondiciones	Tiene que existir al menos un objeto con estado físico asociado para asignarle un volumen de colisión y determinar su matriz de inercia inicial	
Poscondiciones	Quedan calculada la matriz de inercia inicial y asociada al volumen de colisión del cuerpo al que pertenece.	

Tabla 6 Caso de Uso expandido: Actualizar estado del CR

Caso de uso		
CU-5	Actualizar estado del CR	
Propósito	Actualizar el estado del cuerpo rígido en cada iteración de la simulación.	
Actores	Es extensión del CU “Actualizar estado geométrico”	
Resumen: El CU “Actualizar el estado geométrico” solicita actualizar el estado (posición y orientación) de un cuerpo. Si este cuerpo tiene un controlador físico asociado se inicia este caso de uso. Una vez iniciado se invoca al CU-5”Calcular Fuerzas” en el que se calcula la fuerza y el torque resultantes actuando sobre el CR. Con este dato se resuelven todas las ecuaciones del movimiento del cuerpo por alguno de los métodos numéricos a utilizar (Euler, Rungue Kutta, Punto Medio). Se calculan los nuevos valores de las cantidades auxiliares. El caso de uso finaliza cuando se actualiza el estado del cuerpo rígido.		
Referencias	16.1 al 16.5, 17, 18, 19, CU-6 “Calcula fuerza”(include)	
Acción del actor		Respuesta del sistema
1- Se solicita actualizar estado del CR.		2- Con el tiempo anterior en el que se hizo la actualización y el tiempo actual se calcula el tiempo transcurrido desde la actualización anterior.
		3- Se invoca al CU-5 “Calcular Fuerzas” que calcula y aplica la fuerza y el torque resultante al CR.
		4- Se verifica que método de resolución de ecuaciones se va utilizar. a) Si es el método de Euler ir a la “sección 2”. b) Si es el método de Punto Medio ir la “sección 3”. c) Si es el método de Rungue Kutta ir la “sección 4”.
		4- Se calcula el nuevo tensor de inercia.
		5- Se calcula la nueva velocidad lineal.
		6- Se calcula la nueva velocidad angular y el spin.
		7- Se actualiza el estado geométrico global del nodo asociado al controlador físico (posición y orientación).

	7- Se actualiza el tiempo.
Sección 1: Resolver ecuaciones del movimiento del CR.	
	1- Se actualiza la posición del cuerpo con su posición, velocidad anterior y el deltatime.
	2- Se actualiza la orientación del cuerpo con su orientación, velocidad angular, *quaternion* de rotación anterior y el deltatime.
	3- Se actualiza el momento lineal con el momento lineal anterior, la fuerza total incidente sobre el cuerpo y el deltatime.
	4- Se actualiza el momento angular con su momento angular anterior, el torque total incidente sobre el cuerpo y el deltatime.
Sección 1: Método de Euler.	
	1- Ir a la Sección 1.
Sección 2: Método de Punto Medio.	
	1- Se calcula la mitad del deltatime.
	2- Ir a la Sección 1.
	3- Se invoca al CU-5 “Calcular Fuerzas” que calcula y aplica la fuerza y el torque resultante al CR.
	4- Ir a Sección 1.
Sección 3: Método de Rungue Kutta.	
	1- Se calcula la mitad del deltatime.
	2- Se calcula la función f1 con la derivada del vector de estado anterior del CR.
	3- Se calcula la función f2 con f1 y la mitad del deltatime.
	4- Se calcula la función f3 con f2 y la mitad del

	deltatime.
	5- Se calcula la función f4 con f3 y el deltatime.
	6- Se actualiza la derivada del vector de estado del CR (velocidad, spin, fuerza y torque) con la fórmula general del método de Rungue Kutta(y= f1 + (2f2 + 2f3 + f4)/6)
	7- Se actualiza la posición del cuerpo con su posición, velocidad según la fórmula del paso 6 y el deltatime.
	8- Se actualiza la orientación del cuerpo con su orientación anterior, velocidad angular y, *quaternion* de rotación según la fórmula del paso 6 y el deltatime.
	9- Se actualiza el momento lineal con el momento lineal anterior, la fuerza total incidente sobre el cuerpo según fórmula del paso 6 y el deltatime.
	10- Se actualiza el momento angular con su momento angular anterior, el torque total incidente sobre el cuerpo según fórmula del paso 6 y el deltatime.
Fin de Secciones	
Precondiciones	Tiene que existir al menos un objeto con controlador físico en la escena para actualizar su estado.
Poscondiciones	Queda actualizado el estado físico del objeto (posición y orientación) según leyes físicas.

Tabla 7 Caso de Uso expandido: Calcular Fuerzas

Caso de uso		
CU-6	Calcular fuerzas	
Propósito	Calcular las fuerzas incidentes sobre el cuerpo rígido.	
Actores	Es inclusión del CU 5 “Actualizar estado del CR”	
Resumen: El caso de uso es invocado por el CU-4 “Actualizar Estado” para el cálculo de la fuerza y torque total incidente sobre el CR. De acuerdo con el medio y la superficie en las que se encuentra el cuerpo se adicionan los valores de las fuerzas de fricción, resistencia, gravedad y los torques. El caso de uso finaliza cuando se calcula y adiciona la fuerza y el torque resultante al cuerpo.		
Referencias	12.1 al 12.4, 13, 14, 15, 21.5	
Acción del actor		Respuesta del sistema
		1- Si la fuerza de gravedad está activa se invoca al CU-8 “Adicionar fuerza (Sección 3)” para adicionar la fuerza de gravedad al cuerpo calculada con su fórmula.
		2 - Si la fuerza de resistencia del medio está activa ir a Sección1.
		3- Se calcula si el cuerpo esta apoyado o no en alguna de las superficies. Si esta apoyado ir a Sección2.
		4- Con todas las fuerzas calculadas se calcula la fuerza resultante incidiendo sobre el cuerpo.
		5- Se calcula el torque total incidente sobre el cuerpo.
		6- Se adicionan ambas fuerzas al cuerpo.
Sección 1: Calcular y aplicar fuerza resistiva del medio		
		1-Se solicita en cual de los ambientes del mundo físico está el cuerpo.

	2- Se solicita el coeficiente de viscosidad del medio.
	3- Se solicita el valor de la densidad del medio.
	4- Se calcula la fuerza resistiva del medio según su fórmula y se invoca al CU-8 “Adicionar fuerza (Sección 3)” para adicionar la fuerza.
Sección 1: Calcular y aplicar la fuerza de fricción al cuerpo	
	1- Se calcula la fuerza normal y se invoca al CU-8 “Adicionar fuerza (Sección 3)” para adicionar esta fuerza.
	2- Si está actuando la fricción, y el cuerpo esta en reposo, con la fuerza normal y el coeficiente de fricción estático se calcula la fuerza de fricción estática, y se invoca al CU-8 “Adicionar fuerza (Sección 3)” para adicionar esta fuerza.
	3- Si el cuerpo no está en reposo con la fuerza normal y el coeficiente de fricción cinético se calcula la fuerza de fricción cinética, y se invoca al CU-8 “Adicionar fuerza (Sección 3)” para adicionar esta fuerza.
Fin de sección	
Precondiciones	Debe existir algún controlador físico al que se le esté actualizando su estado.
Poscondiciones	Quedan actualizadas la fuerza y el torque total incidente sobre el cuerpo.

Tabla 8 Caso de Uso expandido: Responder a colisión

Caso de uso		
CU-7	Responder a colisión	
Propósito	Separar los cuerpos que colisionaron	
Actores	Es extensión del CU “Gestionar colisión”	
Resumen: El caso de uso se inicia cuando el CU “Gestionar colisión” sí dos cuerpos colisionaron solicita separarlos. Se calcula el impulso necesario para separar los cuerpos, en dependencia de sus velocidades, masas y parámetros de impacto. Se aplica el impulso a cada uno de los cuerpos. El caso de uso finaliza cuando se aplican los valores de los impulsos a cada cuerpo. Se inicia en otro caso, por el mismo CU “Gestionar colisión” cuando se desea responder a la colisión de un cuerpo con un plano. Se calcula el impulso necesario para separar al cuerpo del plano y se le aplica este valor al cuerpo.		
Referencias	20.1 al 20.8	
Acción del actor		Respuesta del sistema
Sección1: Respuesta a colisión Cuerpo-Cuerpo		
1-Solicita separar los cuerpos que colisionaron introduciendo: • Pareja de cuerpos que colisionaron. • Normal a la colisión. • Punto de colisión. • Coeficiente de restitución al choque.		2- Se le asigna el valor del coeficiente de restitución de la colisión.
		3- Se le asigna el vector normal a la colisión.
		4- Se le asigna el valor del punto de colisión.
		5- Se obtienen los datos de los cuerpos que colisionaron.
		6- Se calcula la velocidad relativa antes de la colisión.
		7- Se calcula la magnitud del impulso producto de la colisión.

	8- Se actualiza el momento lineal del cuerpo con la magnitud del impulso.
	9- Se actualiza el momento angular del cuerpo con la magnitud del impulso.
Sección2: Respuesta a colisión Cuerpo-Plano	
1- Solicita separar el cuerpo que colisionó con el plano introduciendo: • Cuerpo que colisionó. • Normal a la colisión. • Punto de colisión. • Coeficiente de restitución al choque.	2- Se le asigna el valor del coeficiente de restitución de la colisión.
	3- Se le asigna el vector normal a la colisión.
	4- Se le asigna el valor del punto de colisión.
	5- Se calcula el nuevo impulso del cuerpo producto a la colisión con el plano.
	7- Se actualiza el momento lineal del cuerpo con la magnitud del impulso.
	8- Se actualiza el momento angular del cuerpo con la magnitud del impulso.
Precondiciones	Dos cuerpos colisionando ó la colisión de un cuerpo con un plano.
Poscondiciones	Quedan separados los objetos que colisionaron.

Tabla 9 Caso de Uso expandido: Adicionar Fuerza

Caso de uso		
CU-8	Adicionar fuerza	
Propósito	Adicionar una fuerza al cuerpo	
Actores	Programador(Inicia), es también un extensión del CU “Calcular fuerzas”	
Resumen: El caso de uso se inicia cuando el programador solicita adicionar una fuerza al cuerpo ya sea de forma permanente o en un instante de la simulación.		
Referencias	22.1 al 22.6	
Acción del actor		Respuesta del sistema
Sección1: Adicionar fuerza permanente sobre el centro de masa del cuerpo.		
1- Solicita adicionar una fuerza permanente al cuerpo sobre su centro de masa, introduciendo: • Vector fuerza		2- Se adiciona la fuerza al cuerpo.
Sección2: Adicionar una fuerza permanente sobre un punto del cuerpo.		
1- Solicita adicionar una fuerza permanente al cuerpo sobre un punto, introduciendo: • Vector fuerza • Punto de aplicación.		2- Se adiciona la fuerza al cuerpo.
		3- Se calcula el torque que esta fuerza produce al cuerpo.
Sección3: Adicionar una fuerza que actúe sobre el cuerpo en un instante de tiempo sobre su centro de masa.		
1- Solicita adicionar una fuerza que actúe sobre el cuerpo en un instante de tiempo sobre su centro de masa, introduciendo:		2- Se adiciona la fuerza al cuerpo.

• Vector fuerza	
Sección4: Adicionar una fuerza que actúe sobre el cuerpo en un instante de tiempo sobre un punto.	
1- Solicita adicionar una fuerza que actúe sobre el cuerpo en un instante de tiempo sobre un punto, introduciendo: • Vector fuerza • Punto de aplicación.	2- Se adiciona la fuerza al cuerpo.
	3- Se calcula el torque que esta fuerza produce al cuerpo.
Sección5: Adicionar torque permanente al cuerpo.	
1- Solicita adicionar un torque permanente al cuerpo, introduciendo: • Vector torque.	2- Se adiciona el torque al cuerpo.
Sección6: Adicionar torque que actúe sobre el cuerpo en un instante de tiempo.	
1- Solicita adicionar un torque que actúe en un instante de tiempo, introduciendo: • Vector torque.	2- Se adiciona el torque al cuerpo.

Precondiciones	Existir al menos un cuerpo al que se le aplique una fuerza.
Poscondiciones	Queda afectado el cuerpo por la fuerza introducida.

2.10 Consideraciones técnicas generales

La herramienta de la cual va a formar parte éste módulo tiene las funcionalidades para el control del tiempo. La misma cuenta además con un “ciclo de escena” que se repite durante toda la simulación, en el cual se hacen todos los cálculos de estados geométricos de los objetos. Es en este ciclo donde se va a acoplar las funcionalidades del módulo a crear, el cual incidirá en el cálculo de dichos estados geométricos de los objetos. El “paso” o intervalo de tiempo para los cálculos estará en correspondencia con este ciclo de escena.

Como la cantidad de procesos concurrentes en la computadora puede influir en el logro del realismo en la escena que se logre mediante la simulación física, se aplicará también la variante en que se pueda mantener un espacio de tiempo fijo definido por el programador para la actualización del estado de cada uno de los cuerpos. Se podrá trabajar de ambas formas, dejando en manos del programador la decisión del método a utilizar.

El conjunto de clases que soporten todas las características que se han señalado en este capítulo será programado en el lenguaje de desarrollo C++. El diseño e implementación se corresponderá con la filosofía de Programación Orientada a Objetos.

Conclusiones

En el presente capítulo se definió el sistema propuesto y quedan sentadas las bases técnicas por las que se regirá. Quedaron establecidos los requisitos funcionales de éste, y descritos los casos de uso que les permitirá a sus futuros usuarios obtener los resultados esperados por el cliente.

3

Capítulo 3 Diseño e Implementación del Sistema

Introducción

En este capítulo se define la estructura del sistema en estudio, siempre basándose en los requisitos funcionales y no funcionales que fueron seleccionados en las etapas anteriores. Se presentan los artefactos involucrados en el diseño del módulo, que serían Diagramas de clases y Diagramas de secuencia de la realización de los casos de uso. También se encuentras los diagramas de componentes en los que se definen como se harán físicas las clases de diseño en el lenguaje seleccionado.

Se especifican además otros aspectos para la comprensión de los diagramas y se describen en tablas las clases de diseño correspondientes al módulo. Se presentan también los patrones de diseño utilizados justificando su elección.

3.1 Diseño

Antes de mostrar los diagramas correspondientes a este flujo de trabajo se aclararan cuestiones importantes que permiten la comprensión de los diagramas de clases y secuencia:

Se hace uso en el diseño de las clases de la nomenclatura usada en la Herramienta “Scene ToolKit”. Ver anexo: Estándares de codificación.

Las clases representadas con fondo sombreado son clases pertenecientes a la herramienta que tienen alguna relación con las clases del módulo, es por eso que para una mayor comprensión de los diagramas se representan aquí.

Los métodos de acceso a miembros (get y set) se omitieron de los Diagramas de Clases para la simplificación de los mismos.

3.1.1 Diagrama de Paquetes de Clases de Diseño

Las clases utilizadas en el módulo están organizadas por paquetes distinguiendo las responsabilidades de cada una de ellas. A continuación se muestran los paquetes en los que se agrupan las clases y las dependencias entre ellos. El paquete “Physics” es el que agrupa las clases a desarrollar, los demás tienen de alguna forma relación con él y forman parte de la herramienta “Secene ToolKit”.

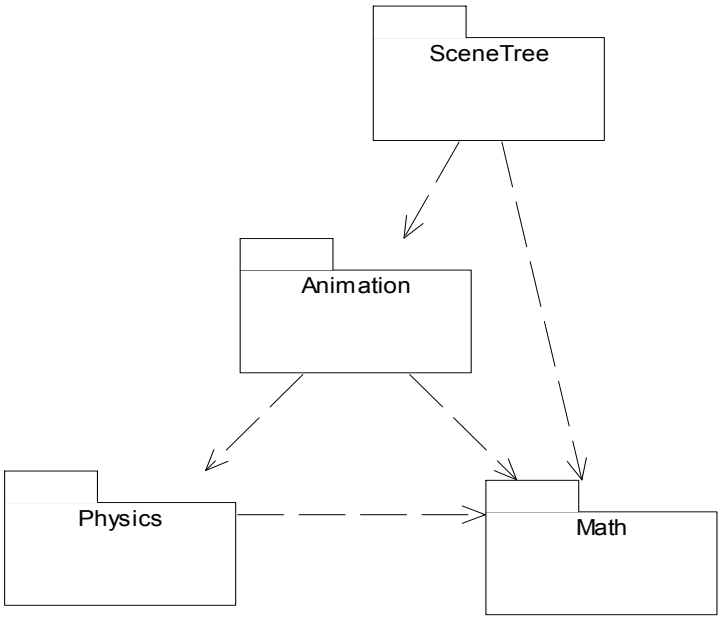


Figura 16 Diagrama de paquetes de clases de diseño

Cada uno de los paquetes contiene las clases especificadas a continuación:

Tabla 10 Relación entre paquetes y clases

Paquetes	Clases contenidas en el paquete utilizadas en el módulo
SceneTree	CNode
Animation	CController
Math	CVector3 CMatrix3 CQuaternion
Physic	CPhysicController CPhyiscState CPhysicWorld CCollisionVolume CCollisionResponse

3.1.2 Diagramas de Clases de Diseño

Para la representación de la relación de las clases se agruparon las mismas por las funcionalidades principales que se atienden en el módulo:

- Propiedades del mundo físico generales.
- Actualizar estado del cuerpo.
- Responder a colisión.

El diagrama de clases del segundo proceso se separó en partes funcionales para lograr representar todas las relaciones de la forma más clara posible.

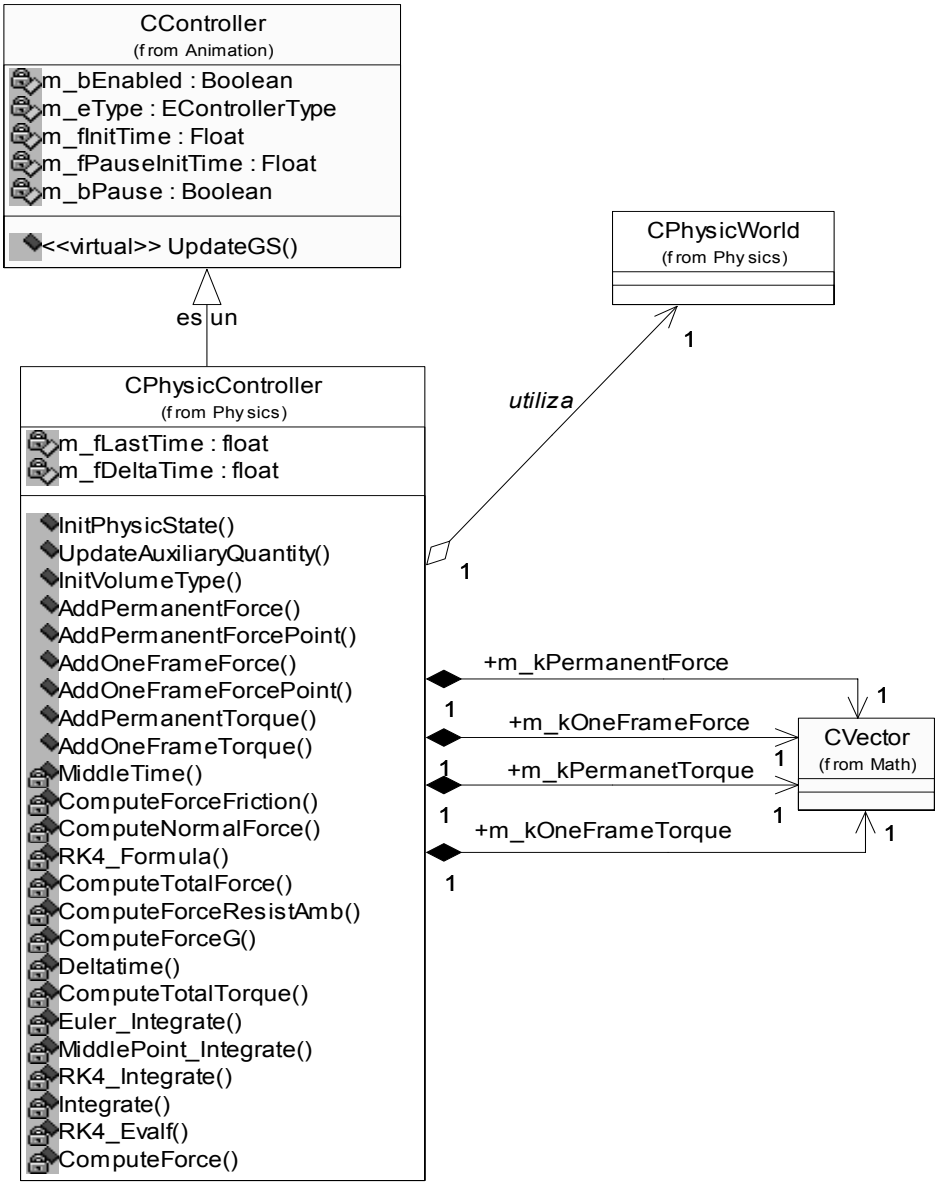


Figura 17 Diagrama de Clases de Diseño: Actualizar estado del CR(A)

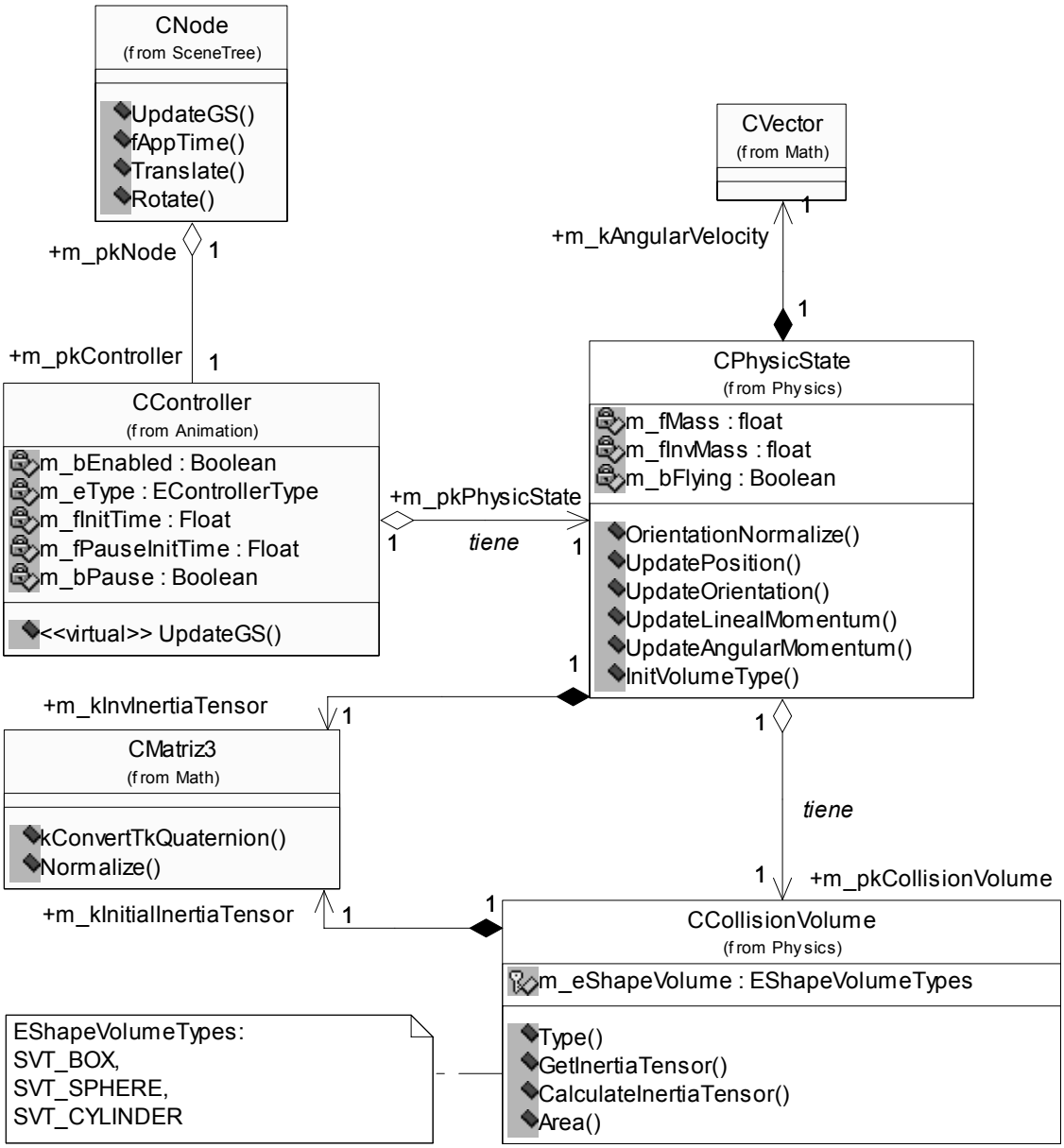


Figura 18 Diagrama de Clases de Diseño: Actualizar estado del CR (B)

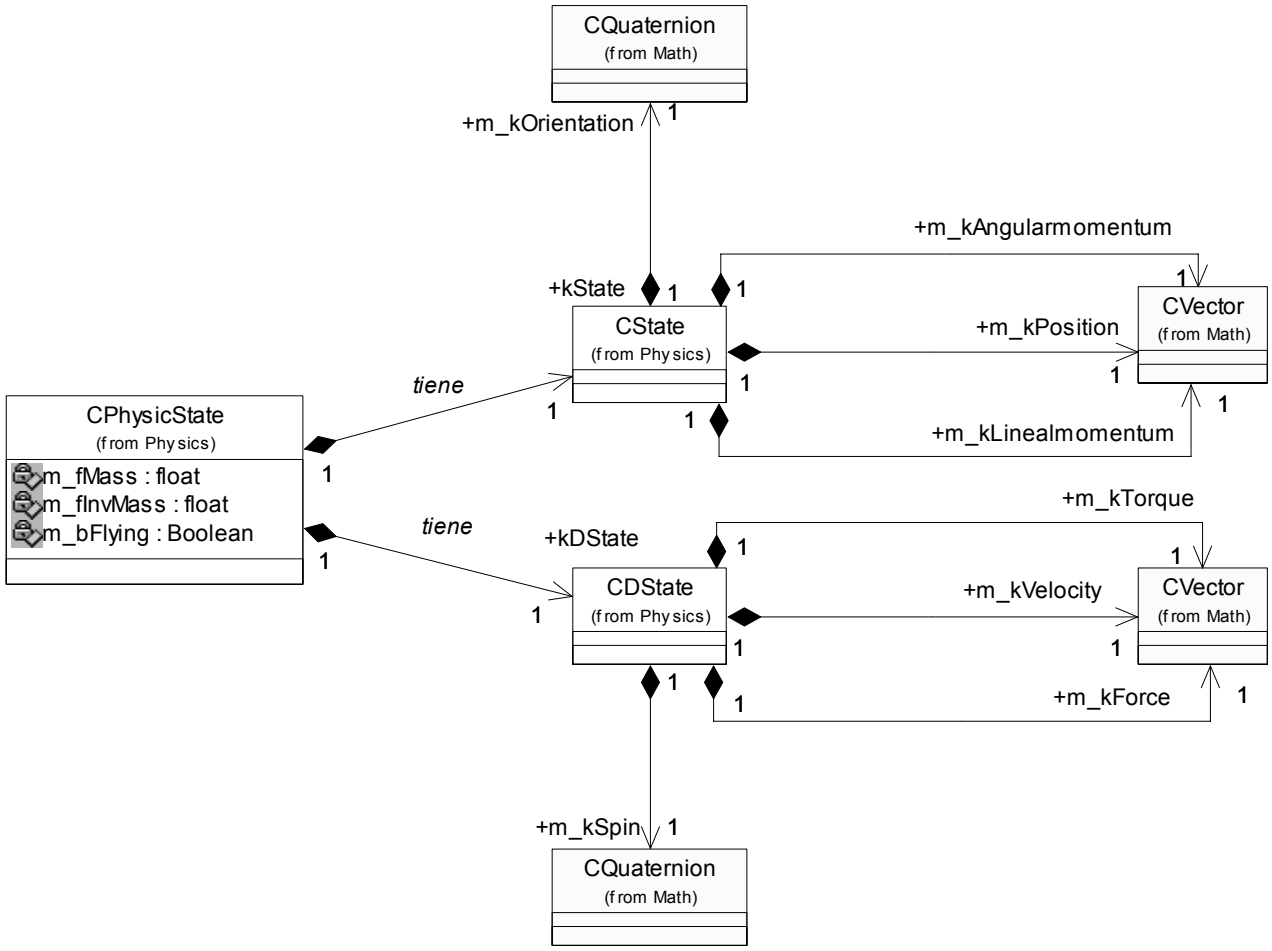


Figura 19 Diagrama de Clases de Diseño: Actualizar estado del CR (C)

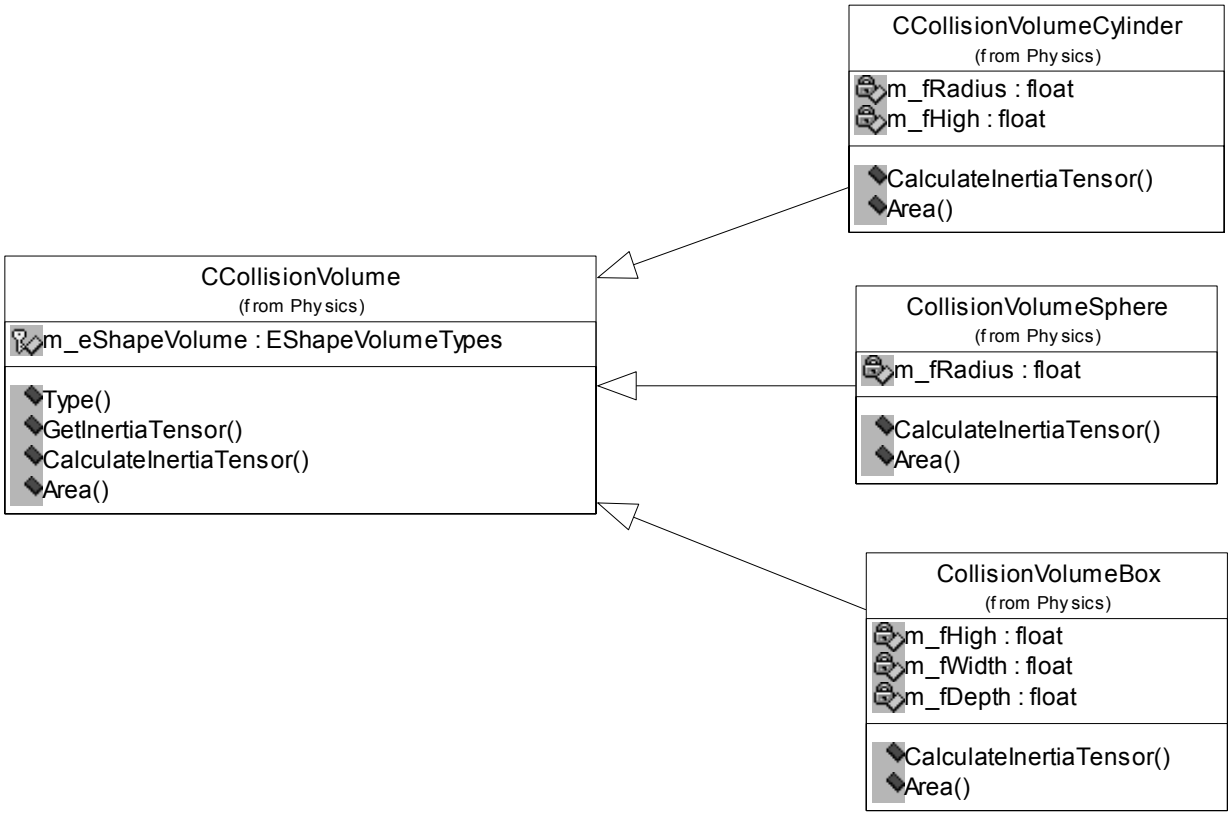


Figura 20 Diagrama de Clases de Diseño: Actualizar estado del CR (D)

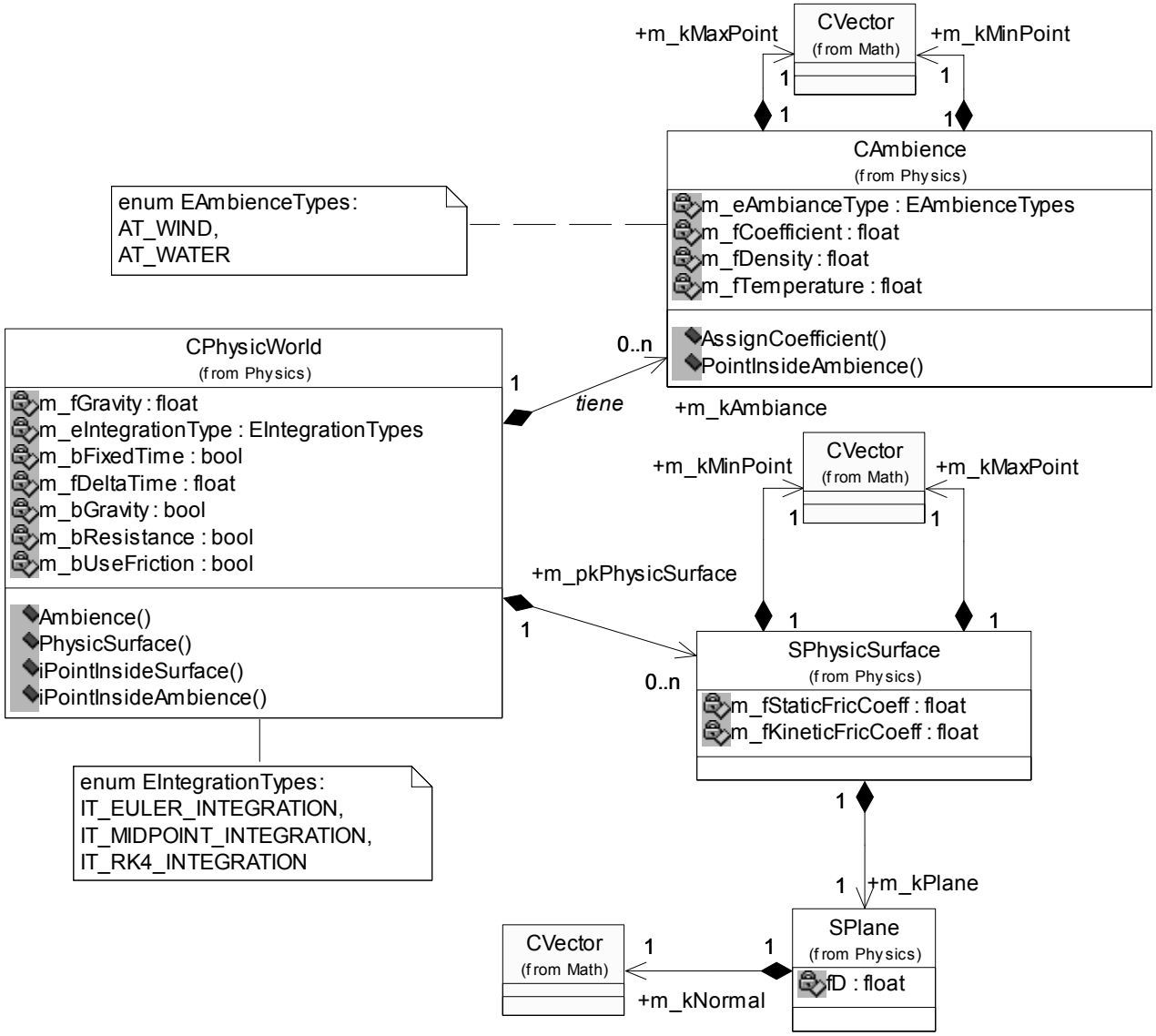


Figura 21 Diagrama de Clases de Diseño: Propiedades del mundo físico

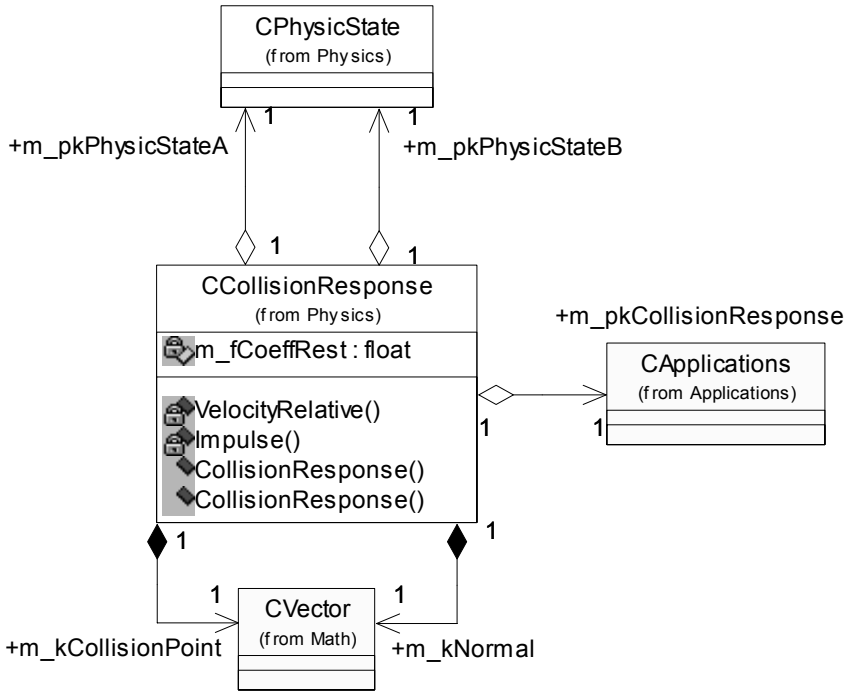


Figura 22 Diagrama de Clases de Diseño: Responder a colisión

3.1.3 Diagramas de interacción (Secuencia)

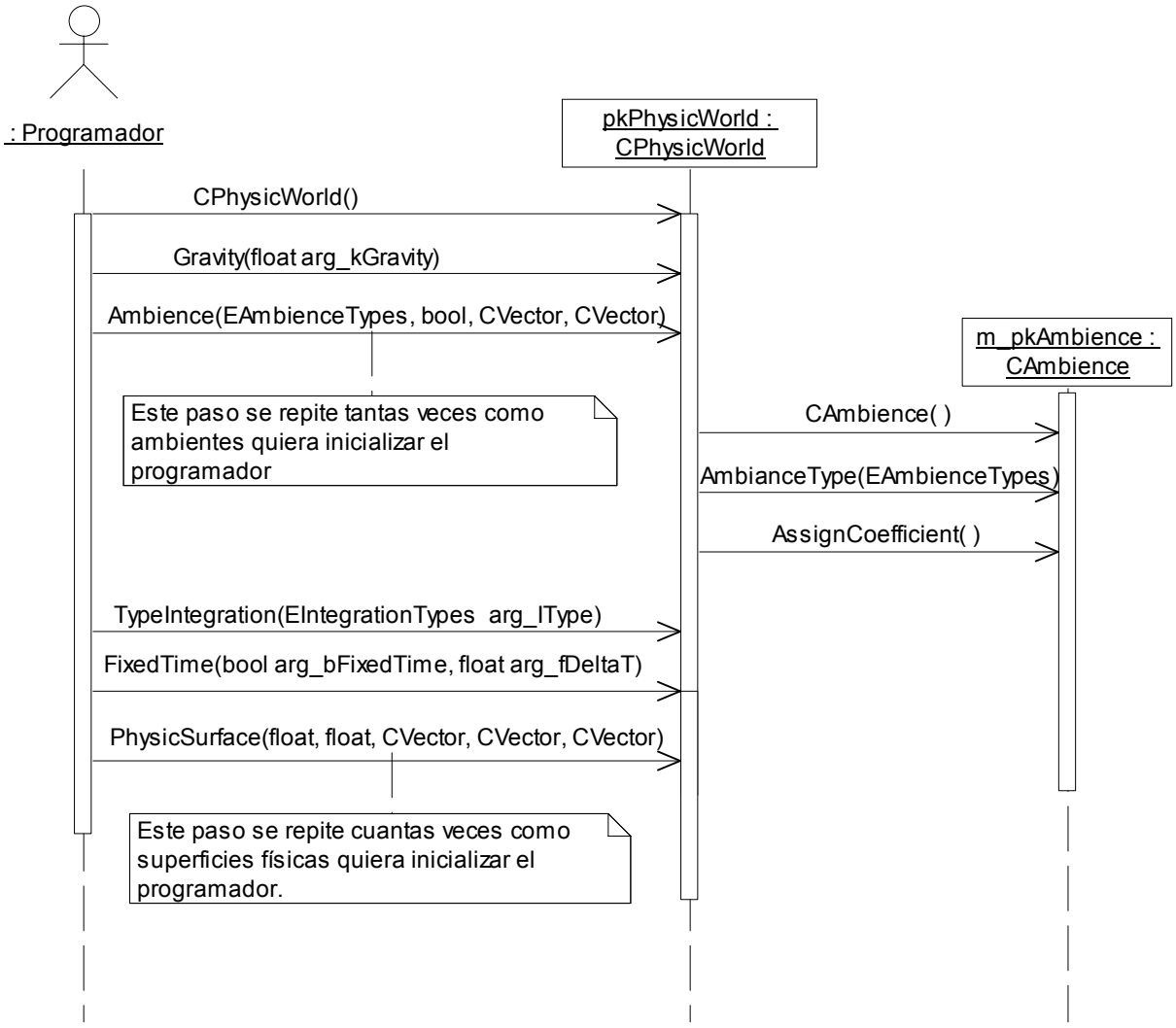


Figura 23 Diagrama de Secuencia: Inicializar mundo físico

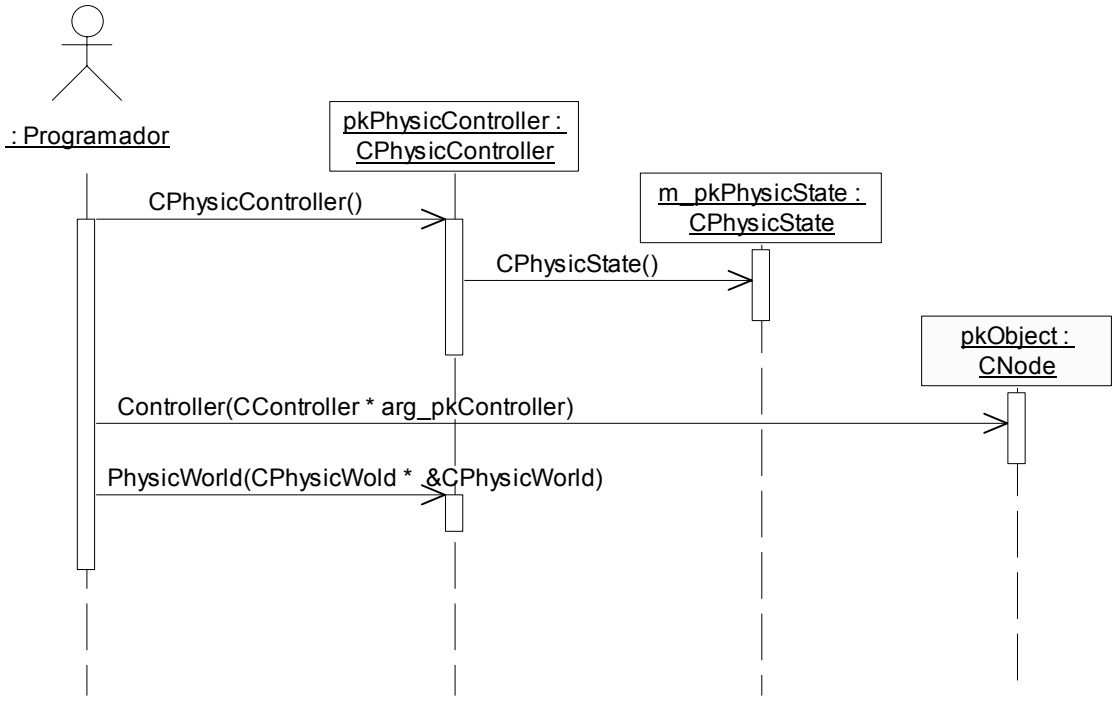


Figura 24 Diagrama de Secuencia: Crear controlador físico

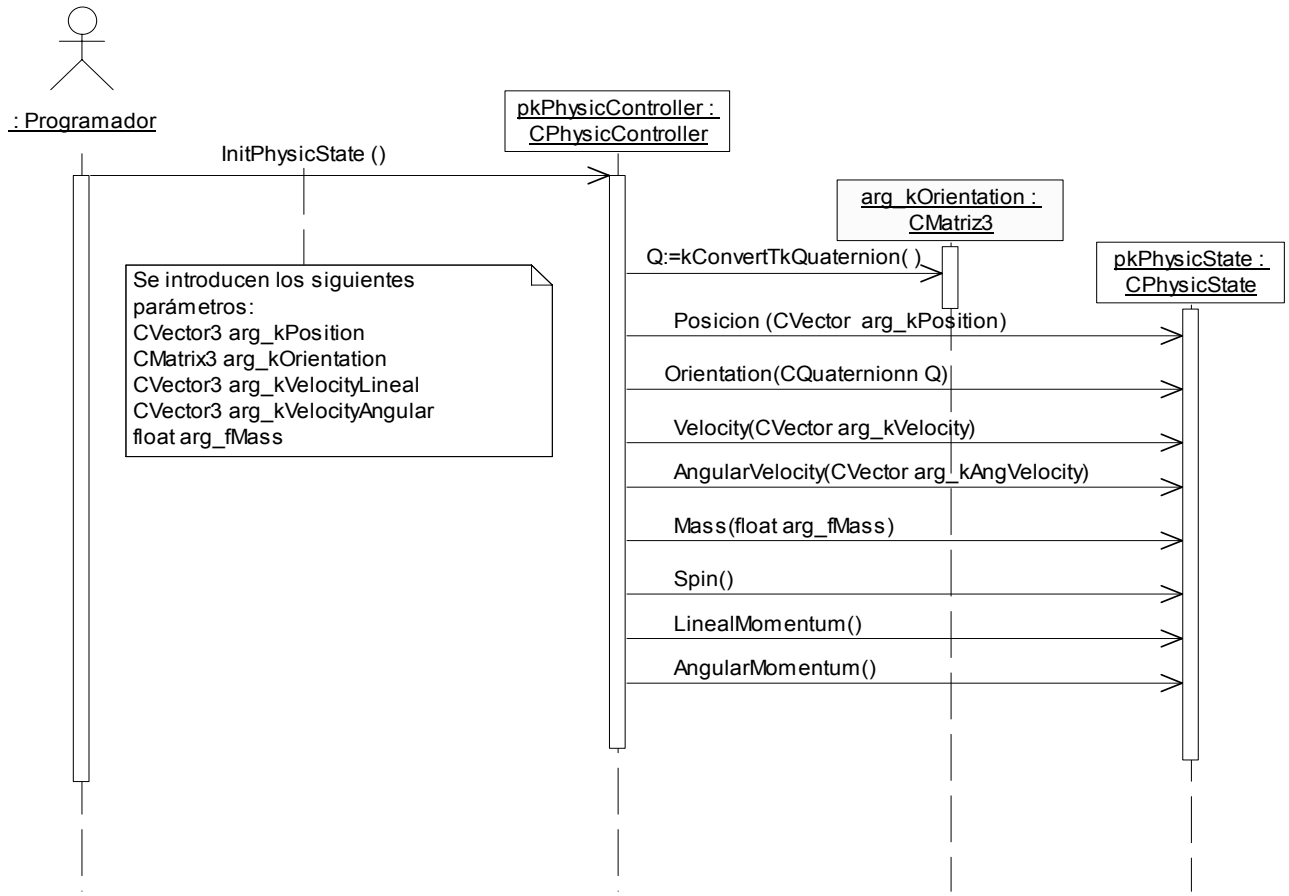


Figura 25 Diagrama de Secuencia: Inicializar condiciones del CR

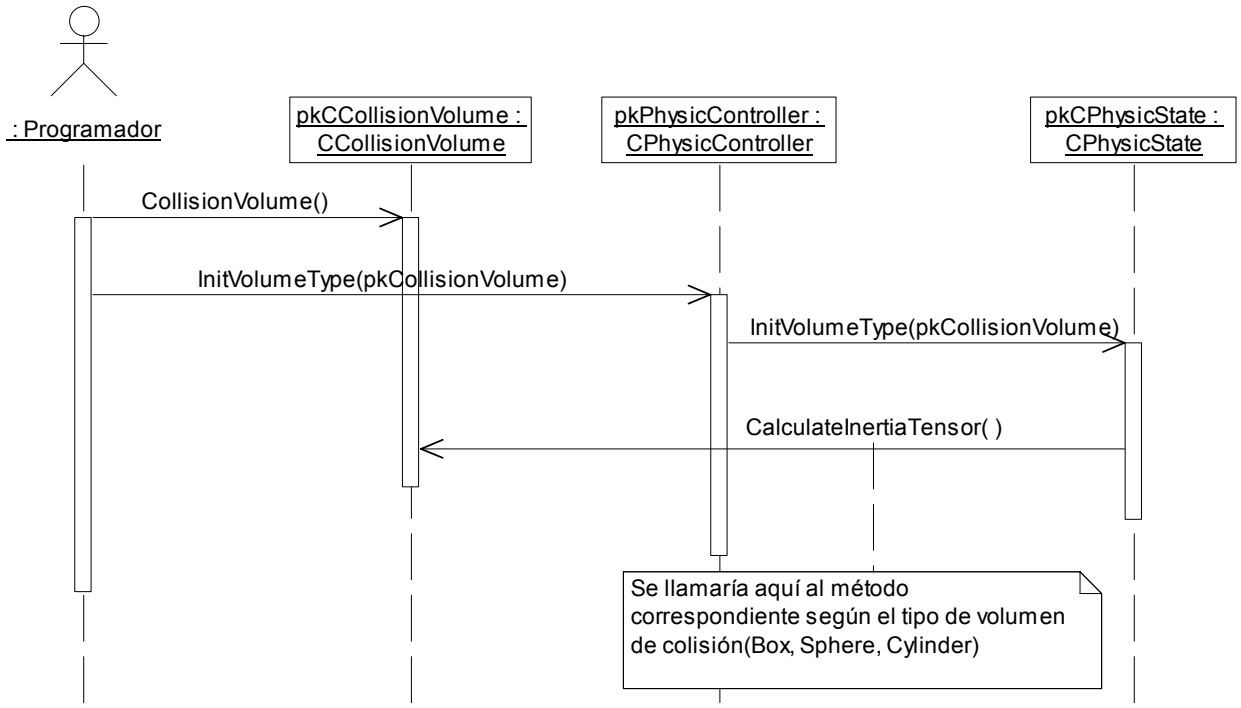


Figura 26 Diagrama de Secuencia: Inicializar volumen de colisión

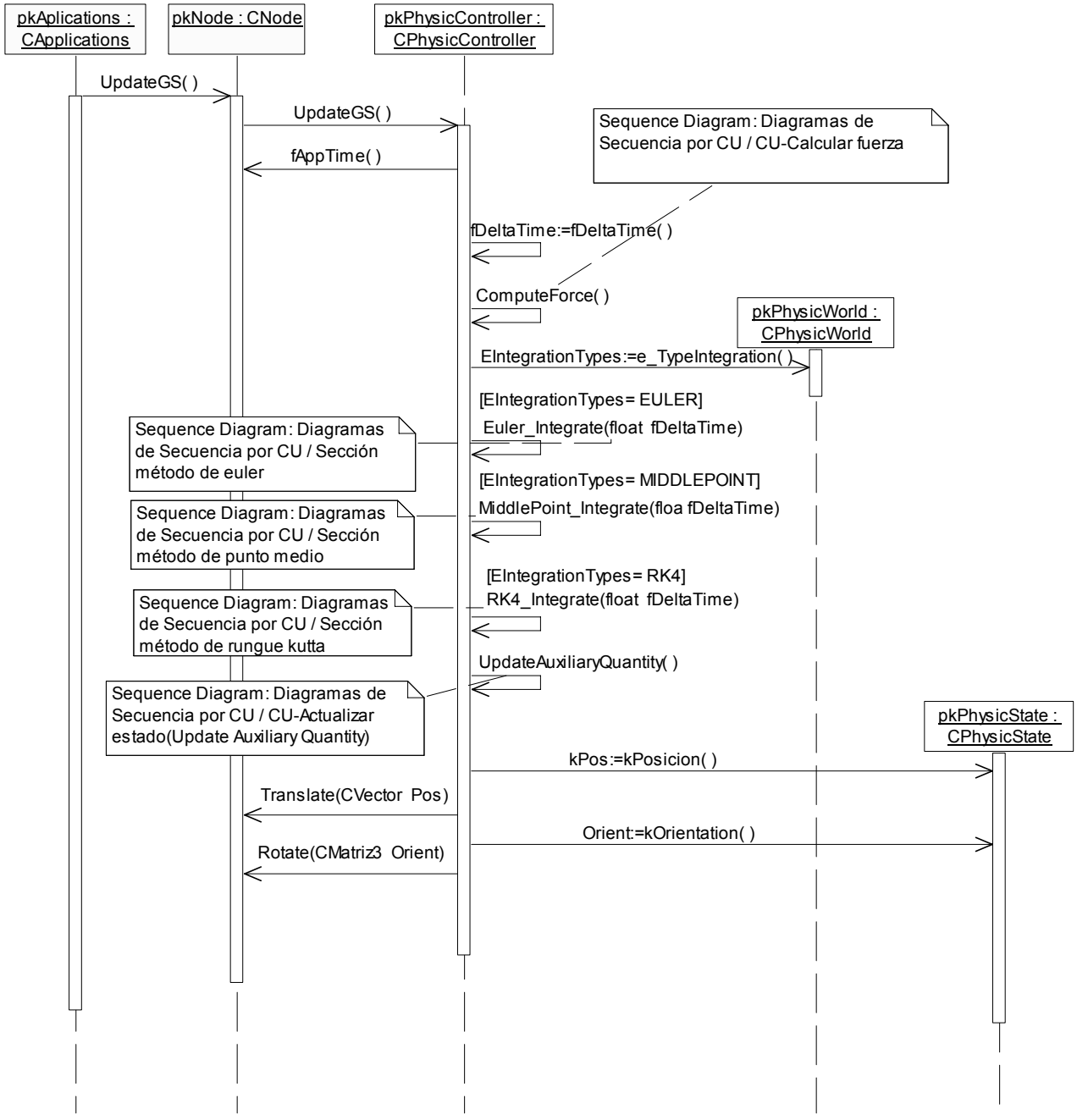


Figura 27 Diagrama de Secuencia: Actualizar estado del CR

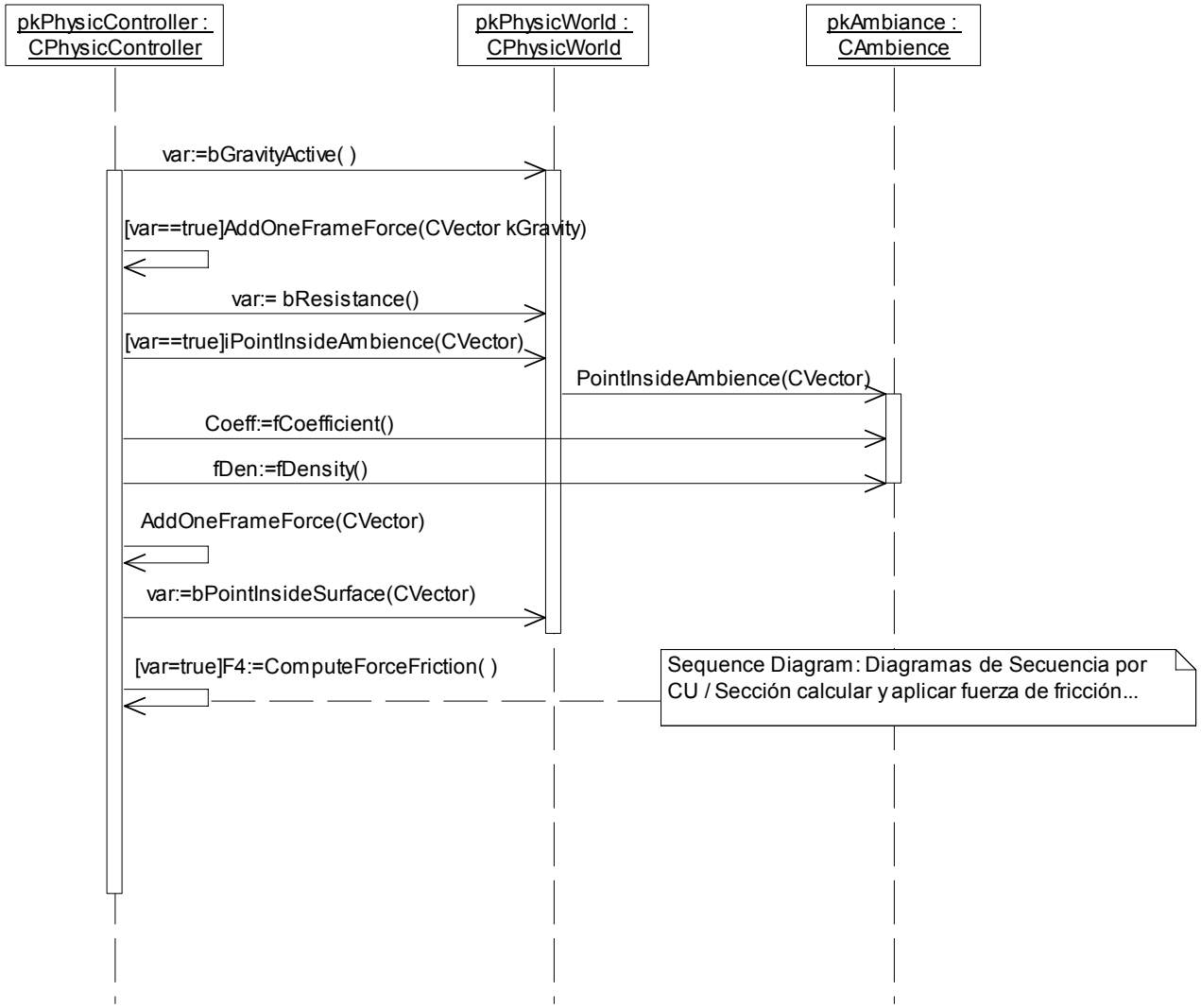


Figura 28 Diagrama de Secuencia: Calcular fuerza

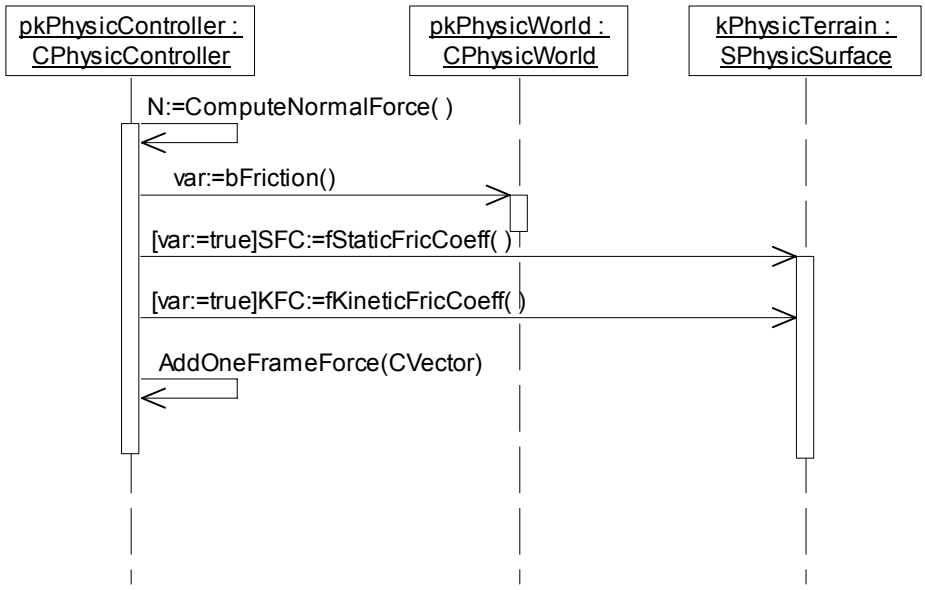


Figura 29 Diagrama de Secuencia: Calcular Fuerza (Sección Fuerzas de fricción y normal)

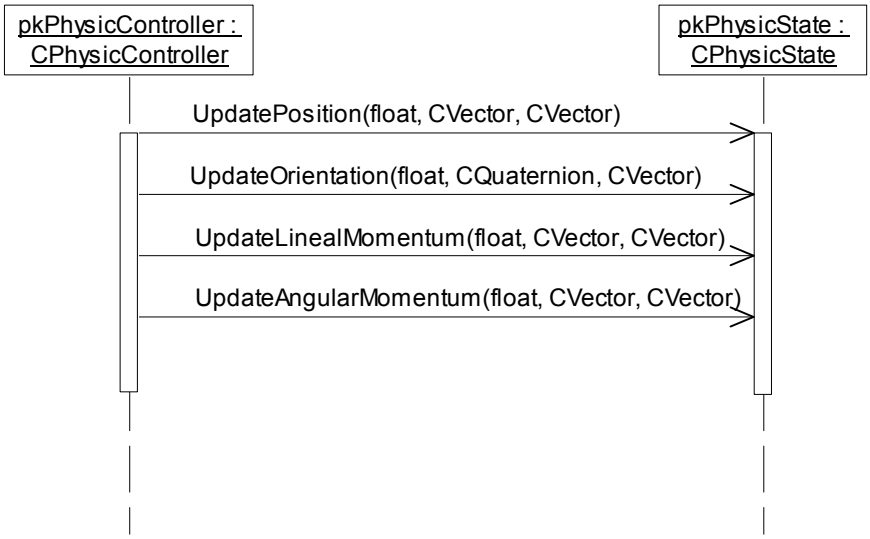


Figura 30 Diagrama de Secuencia: Sección Integrate

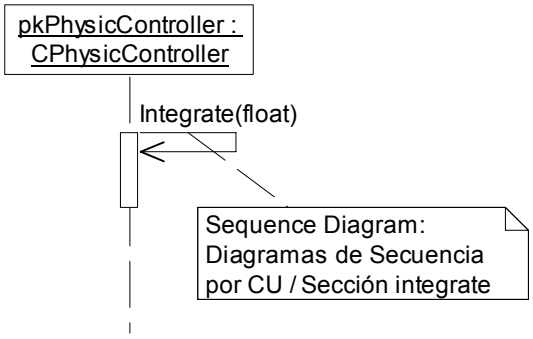


Figura 31 Diagrama de Secuencia: Sección Método de Euler

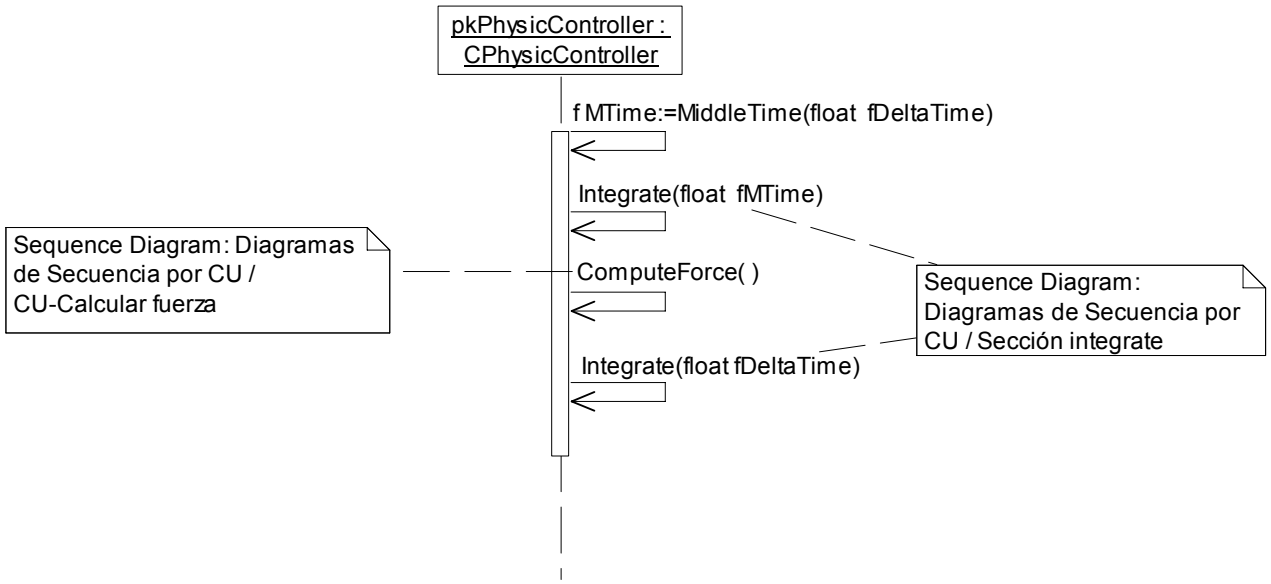


Figura 32 Diagrama de Secuencia: Sección Método de Punto Medio

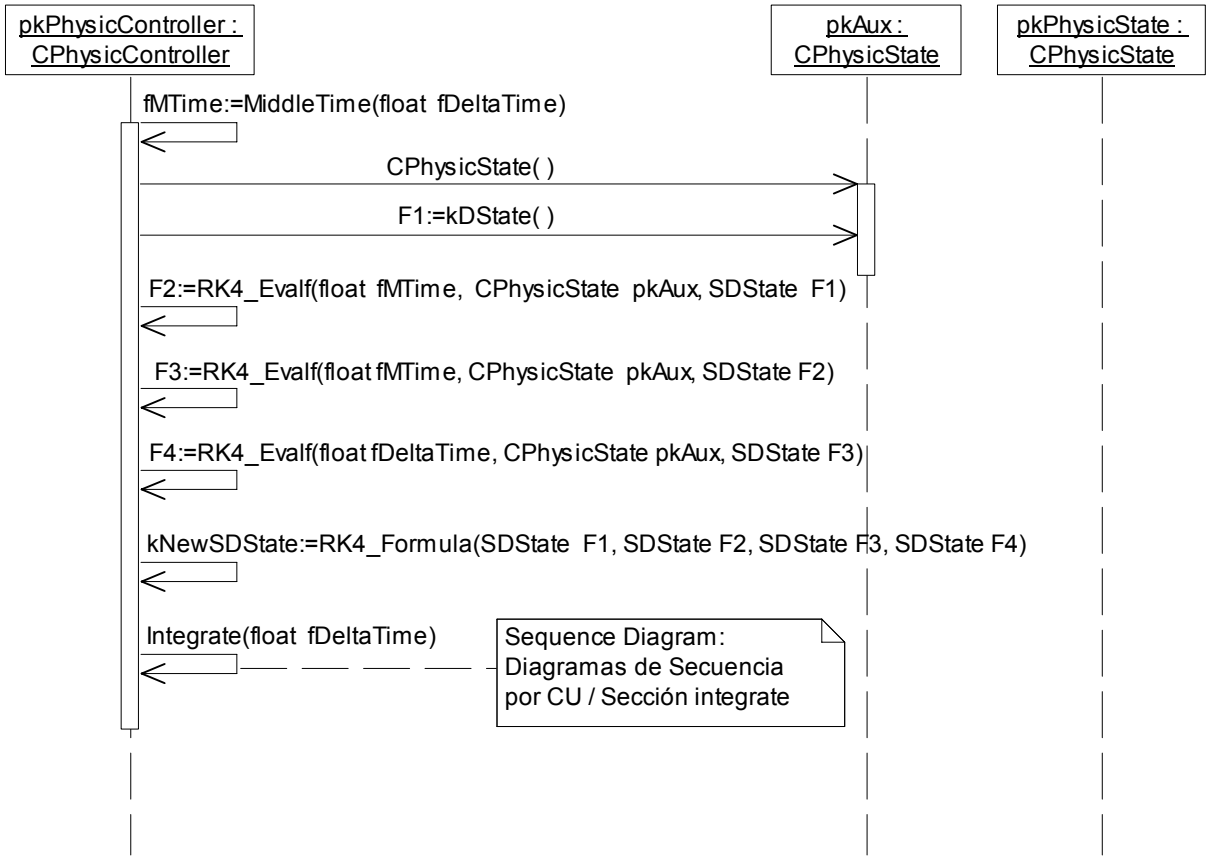


Figura 33 Diagrama de Secuencia: Sección Método de Rungue Kutta 4

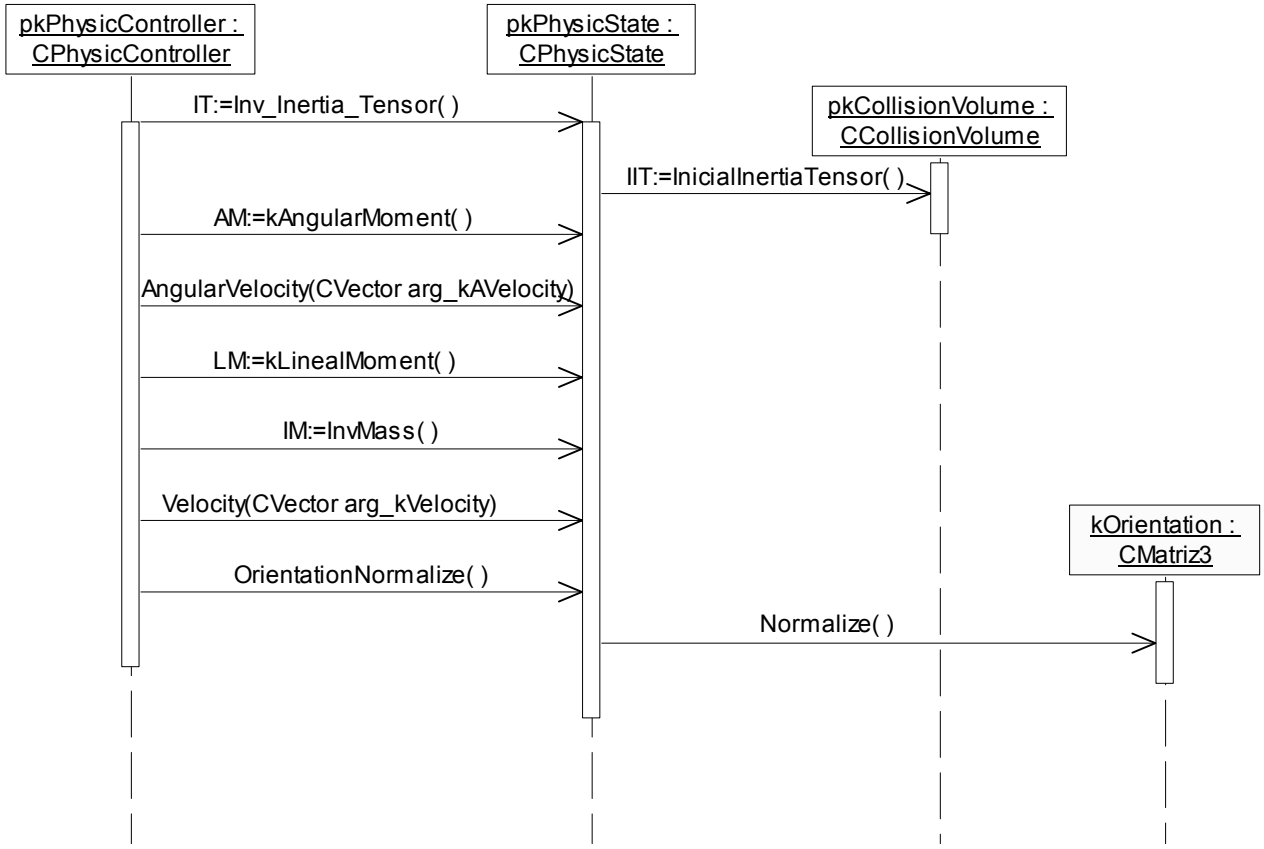


Figura 34 Diagrama de Secuencia: Actualizar cantidades auxiliares

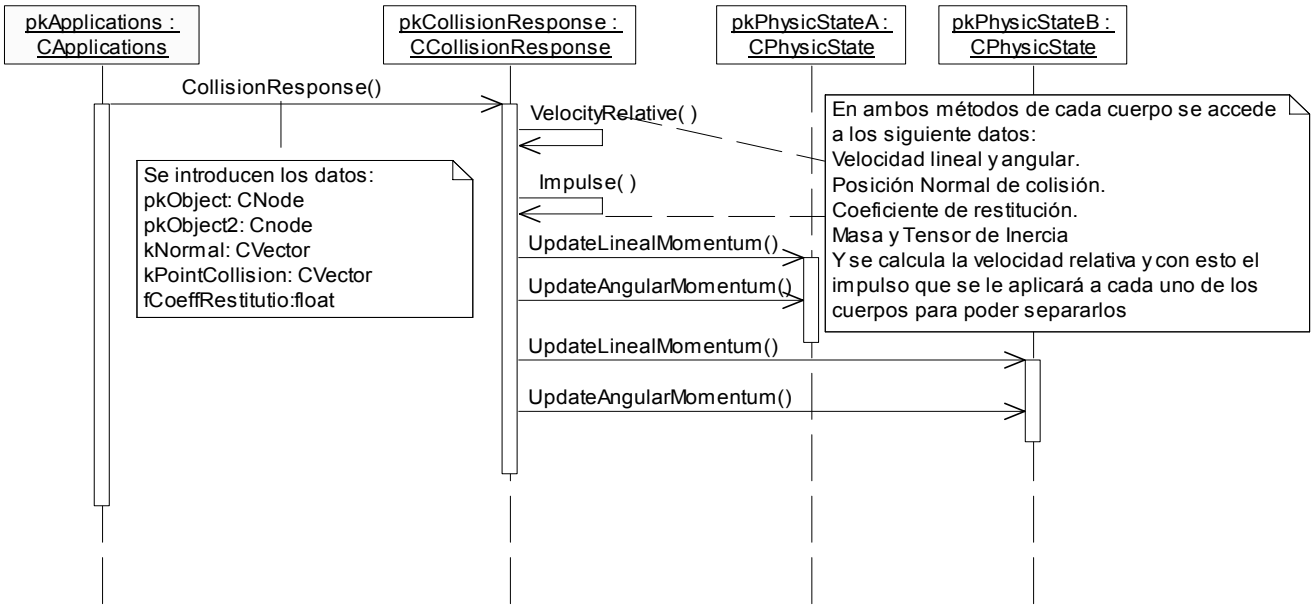


Figura 35 Diagrama se Secuencia: Responder a colisión Cuerpo-Cuerpo

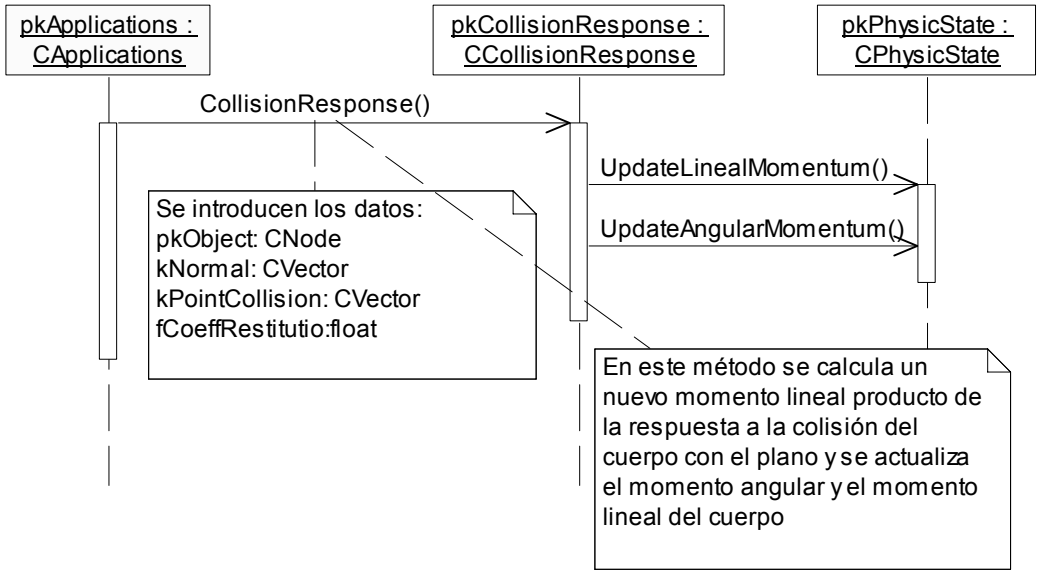


Figura 36 Diagrama de Secuencia: Responder a colisión Cuerpo-Plano

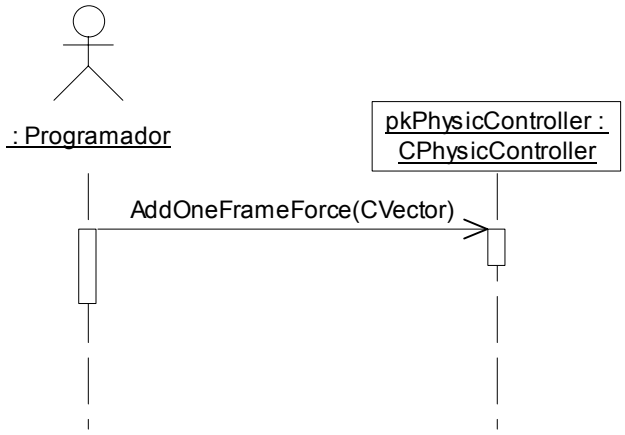


Figura 37 Diagrama de Secuencia: Sección Adicionar Fuerza que actúe en un instante de tiempo

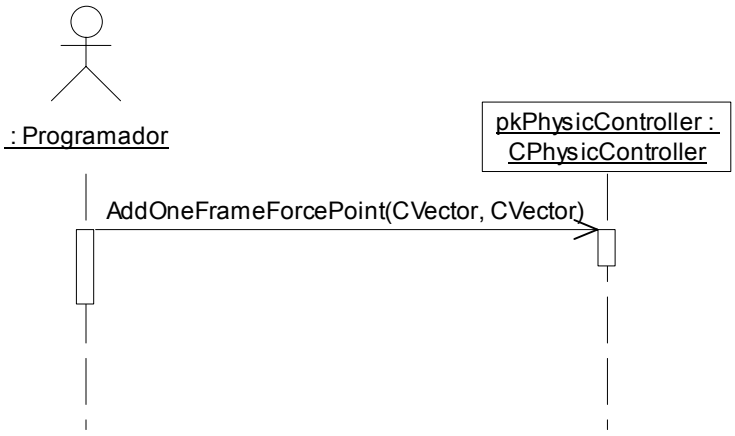


Figura 38 Diagrama de Secuencia: Sección Adicionar Fuerza que actúe en un instante de tiempo en un punto del cuerpo

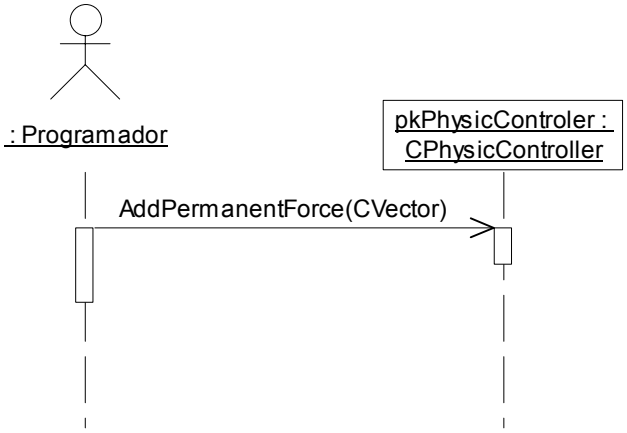


Figura 39 Diagrama de Secuencia: Sección Adicionar Fuerza permanente

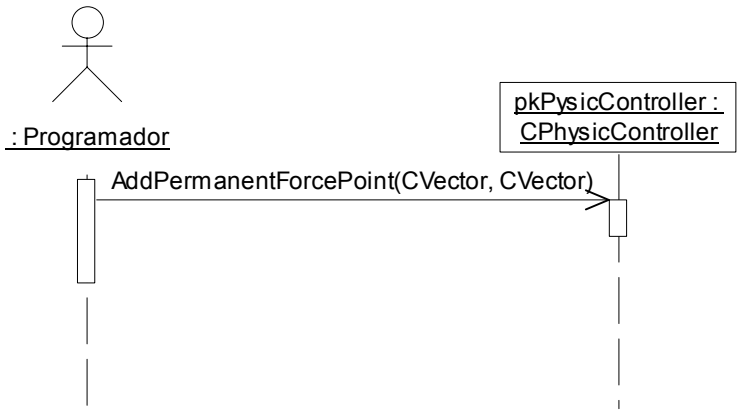


Figura 40 Diagrama de Secuencia: Sección Adicionar Fuerza permanente que actúe en un punto del cuerpo

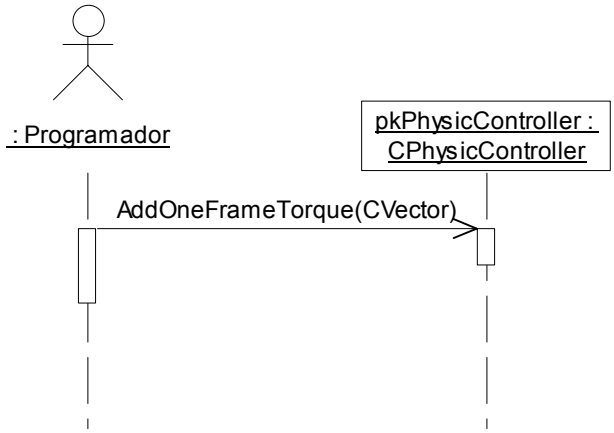


Figura 41 Diagrama de Secuencia: Sección Adicionar Torque que actúe en un instante de tiempo

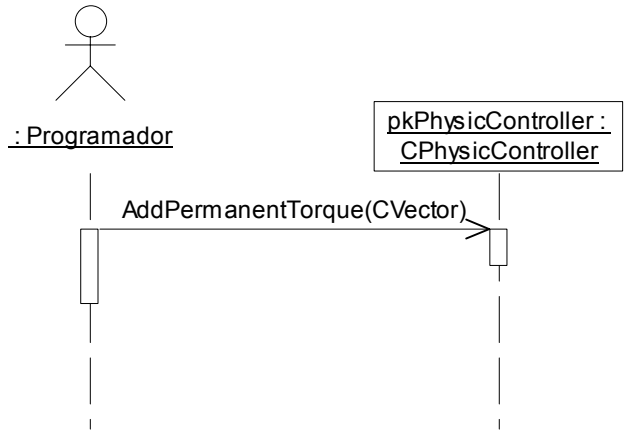


Figura 42 Diagrama de Secuencia: Sección Adicionar Torque permanente

3.2 Descripción de las clases

Nombre: CAmbience	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_eAmbienceType	EAmbienceTypes
m_fCoefficient	float
m_fTemperature	float
m_fDensity	float
m_kMinPoint	CVector
m_kMaxPoint	CVector
Responsabilidad 1:	
Nombre:	AssignCoefficient
Descripción:	Según el tipo de ambiente en el que se encuentre se le asigna el valor del coeficiente, de la densidad y la temperatura.
Responsabilidad 2:	
Nombre:	PointInsideAmbience(CVector arg_kPoint)
Descripción:	Dado un punto(x,y,z) devuelve si ese punto pertenece al volumen que contiene al ambiente.

Nombre: SPlane(Struct)	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_kNormal	CVector
m_fD	float

Nombre: SPhysicSurface(Struct)	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_fStaticFricCoeff	float
m_fKineticFricCoeff	float
m_kPlane	SPlane
m_kMinPoint	CVector
m_kMaxPoint	CVector

Nombre: CPhysicWorld	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_eIntegrationType	EIntegrationTypes
m_bFixedTime	bool
m_fDeltaTime	float
m_bGravityActive	bool
m_fGravity	float
m_bResistance	bool
m_kAmbience	vector<CAmbience>
m_bUseFriction	bool
m_kPhysicSurfaces	vector<SPhysicSurface>
m_iCountSurfaces	int
Responsabilidad 1:	
Nombre:	PhysicSurface(float arg_SFC, float arg_KFC,CVector3 arg_Normal, CVector3 arg_Point,CVector arg_kMinPoint,CVector arg_kMaxPoint)
Descripción:	Adiciona una superficie a la lista que las almacena y le asigna los coeficientes de fricción estática y cinética, y con el valor de la normal del plano y uno de los puntos que lo delimitan calcula el coeficiente D de la ecuación del plano, asigna también las dimensiones del plano que contiene la superficie dadas por el punto mínimo y el punto máximo de éste.
Responsabilidad 2:	
Nombre:	iPointInsideSurface(CVector3 arg_kPoint)
Descripción:	Comprueba si un punto está a una distancia mínima de una superficie (d<0.001) y si se encuentra dentro de las dimensiones de esta, si es así se puede considerar que el punto está en la superficie y devuelve la posición en la lista de superficies de la superficie que cumple estas condiciones.
Responsabilidad 3:	
Nombre:	Ambience(EAmbienceTypes arg_eAmbienceType, bool arg_bResistance, CVector3 arg_kPointMin, CVector3 arg_kPointMax)
Descripción:	Adiciona un ambiente a la lista que los almacena, asignando el tipo de ambiente y si se va a utilizar la resistencia del mismo como fuerza influyente en los cuerpos, asigna también las dimensiones del volumen que va a contener el ambiente

	dadas por el punto mínimo y el punto máximo de esta.
Responsabilidad 4:	
Nombre:	iPointInsideAmbience(CVector3 m_kPoint)
Descripción:	Comprueba si un punto está contenido en el volumen de algún ambiente, si es así se puede considerar una fuerza del ambiente que afecta al objeto, se devuelve la posición en la lista del ambiente del que cumpla estas condiciones

Nombre: SState(Struct)	
Tipo de clase: Entidad	
Atributo	**Tipo**
kPosition	CVector
kOrientation	CQuaternion
kAngularMomentum	CVector
kLinealMomentum	CVector

Nombre: SDState(Struct)	
Tipo de clase: Entidad	
Atributo	**Tipo**
kVelocity	CVector
kSpin	CQuaternion
kForce	CVector
kTorque	CVector

Nombre: CPhysicState	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_kState	SState
m_kDState	SDState
m_fMass	float
m_fInvMass	float
m_kAngularVelocity	CVector3
m_kInertiaTensor	CMatrix3
m_kInvInertiaTensor	CMatrix3
m_bFlying	bool
m_kCollisionVolume	CCollisionVolume
Responsabilidad 1:	
Nombre:	kInertiaTensor()
Descripción:	Calcula y devuelve la matriz del tensor de inercia actual del cuerpo con la orientación actual y la matriz de inercia inicial.
Responsabilidad 2:	

Nombre:	kInvInertiaTensor()
Descripción:	Calcula y devuelve la inversa de la matriz del tensor de inercia actual del cuerpo con la orientación actual y la inversa de la matriz de inercia inicial.
Responsabilidad 3:	
Nombre:	InitVolumeType(CCollisionVolume * &arg_kCollisionVolume)
Descripción:	Inicializa el volumen de colisión según su tipo y calcula la matriz de inercia inicial.

Nombre: CPhysicController	
Tipo de clase: Control	
Atributo	**Tipo**
m_fLastTime	float
m_pkPhysicWorld	CPhysicWorld
m_kPermanentForce	CVector
m_kOneFrameForce	CVector
m_kPermanentTorque	CVector
m_kOneFrameTorque	CVector
m_pkPhysicState	CPhysicState
Responsabilidad 1:	
Nombre:	InitPhysisState(CVector3 arg_kPosition,CMatrix3 arg_kOrientation, CVector3 arg_kVelocity,CVector3 arg_kAngularVelocity,float arg_fMass)
Descripción:	Inicializa el estado físico del cuerpo.
Responsabilidad 2:	
Nombre:	UpdateGS()
Descripción:	Es el método de responsabilidad principal en la actualización del estado (posición y orientación) del cuerpo. Con el tiempo que transcurrió desde la última actualización hasta el momento actual solicita actualizar el estado del cuerpo y actualiza la posición y orientación del nodo que lo contiene.
Responsabilidad 3:	
Nombre:	Deltatime()
Descripción:	Calcula el tiempo transcurrido desde la última actualización del estado del cuerpo hasta la actualización actual.
Responsabilidad 4:	
Nombre:	Run_Physic_Simulation(float deltatime)
Descripción:	Solicita actualizar las fuerzas que actúan sobre el cuerpo, con el tiempo y según el método de integración solicita actualizar el estado del cuerpo.
Responsabilidad 5:	
Nombre:	Euler_Integrate(float deltatime)
Descripción:	Según el método de Euler actualiza el estado del cuerpo.
Responsabilidad 6:	
Nombre:	MiddlePoint_Integrate(float deltatime)
Descripción:	Según el método de Punto Medio actualiza el estado del cuerpo.
Responsabilidad 7:	
Nombre:	RK4_Integrate(float deltatime)

Descripción:	Según el método de Rungue-Kutta de orden 4 actualiza el estado del cuerpo.
Responsabilidad 8:	
Nombre:	RK4_Evalf(float deltatime, CPhysicState *aux1, SDState &aux)
Descripción:	Evalúa las distintas funciones necesarias en el método de Rungue-Kutta
Responsabilidad 9:	
Nombre:	UpdateAuxiliaryQuantity()
Descripción:	Actualiza las cantidades auxiliares que definen el estado del cuerpo (velocidad lineal y angular y derivada del *quaternion* de rotación actual) y normaliza la orientación del cuerpo.
Responsabilidad 10:	
Nombre:	ComputeForce()
Descripción:	Calcula todas las fuerzas que actúan sobre el cuerpo (gravedad, normal, fricción, resistencia al medio, fuerza de usuario) y obtiene la fuerza resultante incidiendo sobre el mismo.
Responsabilidad 11:	
Nombre:	ComputeTotalTorque()
Descripción:	Calcula los torques incidentes sobre el cuerpo de las distintas fuerzas que los producen y obtiene el torque resultante.
Responsabilidad 12:	
Nombre:	AddPermanentForce(CVector3 arg_kForce)
Descripción:	Adiciona una fuerza de forma permanente al cuerpo que actúa en la posición de su centro de masa.
Responsabilidad 13:	
Nombre:	AddPermanentForce(CVector3 arg_kForce, CVector3 arg_kPoint)
Descripción:	Adiciona una fuerza permanente al cuerpo que actúa sobre un punto del cuerpo distinto a la posición de su centro de masa, por lo que se calcula el torque que produce esta fuerza y se adiciona al torque total del cuerpo.
Responsabilidad 14:	
Nombre:	AddOneFrameForce(CVector3 arg_kForce)
Descripción:	Adiciona una fuerza que actúa sobre el cuerpo, en un instante de tiempo sobre la posición de su centro de masa.
Responsabilidad 12:	
Nombre:	AddOneFrameForce(CVector3 arg_kForce, CVector3 arg_kPoint)
Descripción:	Adiciona una fuerza que actúa sobre cuerpo, en un instante y en un punto del cuerpo distinto a la posición de su centro de masa, porque lo que se calcula el torque que produce esta fuerza y se adiciona al torque total del cuerpo.
Responsabilidad 12:	
Nombre:	AddPermanentTorque(CVector3 arg_kTorque)
Descripción:	Adiciona un torque permanente al cuerpo.
Responsabilidad 12:	
Nombre:	AddOneFrameTorque(CVector3 arg_kTorque)
Descripción:	Adiciona un torque al cuerpo que sólo actúa en un instante de tiempo.

Nombre: CCollisionVolume	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_eShapeVolume	EShapeVolumeTypes
m_kInitialInertiaTensor	CMatrix3
m_kInitialInvInertiaTensor	CMatrix3
Responsabilidad 1:	
Nombre:	CalculateInertiaTensor(float arg_fMasa)=0
Descripción:	Método virtual puro que se implementa en las clases hijas, de forma general calcula la matriz de inercia del cuerpo con su masa y sus dimensiones.

Nombre: CCollisionVolumeBox	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_fHigh	float
m_fWidth	float
m_fDepth	float
Responsabilidad 1:	
Nombre:	CalculateInertiaTensor(float arg_fMasa)
Descripción:	Calcula según las dimensiones del cuerpo y la masa la matriz de inercia inicial.

Nombre: CCollisionVolumeSphere	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_fRadius	float
Responsabilidad 1:	
Nombre:	CalculateInertiaTensor(float arg_fMasa)
Descripción:	Calcula según las dimensiones del cuerpo y la masa la matriz de inercia inicial.

Nombre: CCollisionVolumeCylinder	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_fRadius	float
m_fHigh	float
Responsabilidad 1:	
Nombre:	CalculateInertiaTensor(float arg_fMasa)
Descripción:	Calcula según las dimensiones del cuerpo y la masa la matriz de inercia inicial.

Nombre: CCollisionResponse	
Tipo de clase: Entidad	
Atributo	**Tipo**
m_pkPhysicStateA	CPhysicState
m_pkPhysicStateB	CPhysicState
m_kCollisionPoint	CVector3
m_kNormal	CVector3
m_fCoeffRest	float
Responsabilidad 1:	
Nombre:	RelativeVelocity()
Descripción:	Calcula la velocidad relativa entre los dos cuerpos que colisionaron.
Responsabilidad 2:	
Nombre:	Impulse()
Descripción:	Calcula con la velocidad relativa el impulso que producto a la colisión se le aplicará a los cuerpos que colisionaron.
Responsabilidad 3:	
Nombre:	CollisionResponse(CNode *arg_pkNodeA, CNode *arg_pkNodeB, CVector3 arg_kCollisionPoint, CVector3 arg_kNormal, float arg_fCoeffRest)
Descripción:	Responde a la colisión entre dos cuerpos aplicando el impulso correspondiente a cada uno de ellos variando el momento angular y lineal y con esto las velocidades angular y lineal respectivamente.
Responsabilidad 4:	
Nombre:	CollisionResponse(CNode *arg_pkNodeA,CVector3 arg_kCollisionPoint,CVector3 arg_kNormal, float arg_fCoeffRest)
Descripción:	Responde a la colisión entre en cuerpo y un plano, aplicando un impulso al cuerpo producto al choque con el plano que produce un cambio en el momento lineal del cuerpo y con esto un cambio en la velocidad.

3.3 Patrones utilizados

“En la terminología de objetos, el patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas(...)” [26]

Es el diseño del módulo se tuvieron en cuenta principalmente los patrones Experto, y Creador. El primero plantea que siempre se debe asignar una responsabilidad al experto en información, o sea, la clase con toda la información necesaria para llevarla a cabo. El segundo expresa que la responsabilidad de crear una instancia de una determinada clase debe asignarse a otra clase, siempre que esta agregue, contenga,

registre o utilice específicamente los objetos de aquella. En este caso la clase experta y creadora del proceso de actualizar el estado de los cuerpos es CPhysicController.

3.4 Diagrama de componentes

En el siguiente diagrama se representa la relación existente entre el Módulo Físico y la Herramienta a la que se incorpora este módulo. Este vínculo se establece por la estrecha relación que existe entre los componentes de la herramienta y los componentes del módulo en desarrollo.

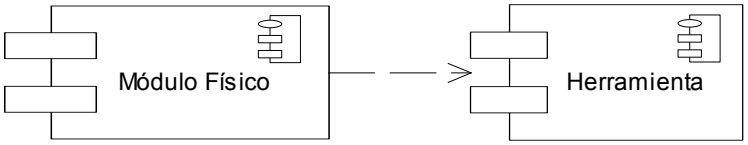


Figura 43 SubPaquetes de Componentes de la vinculación del módulo con la herramienta SceneToolKit

Los paquetes de la herramienta que se muestran a continuación son los que se vinculan con el módulo.

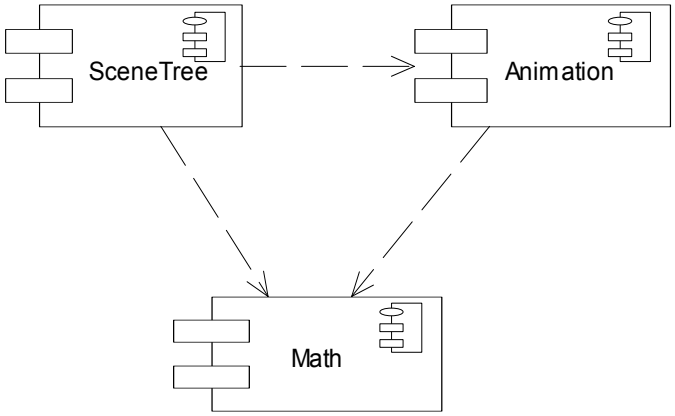


Figura 44 Diagrama de Paquetes de Componentes de la herramienta utilizadas en el módulo

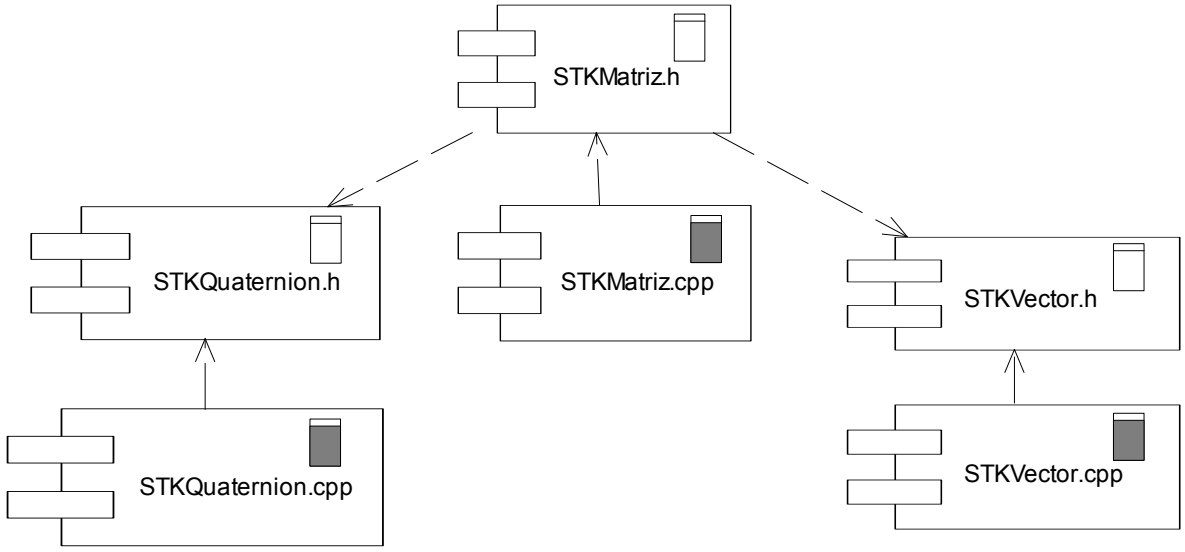


Figura 45 Diagrama de Componentes del paquete matemático (Math)

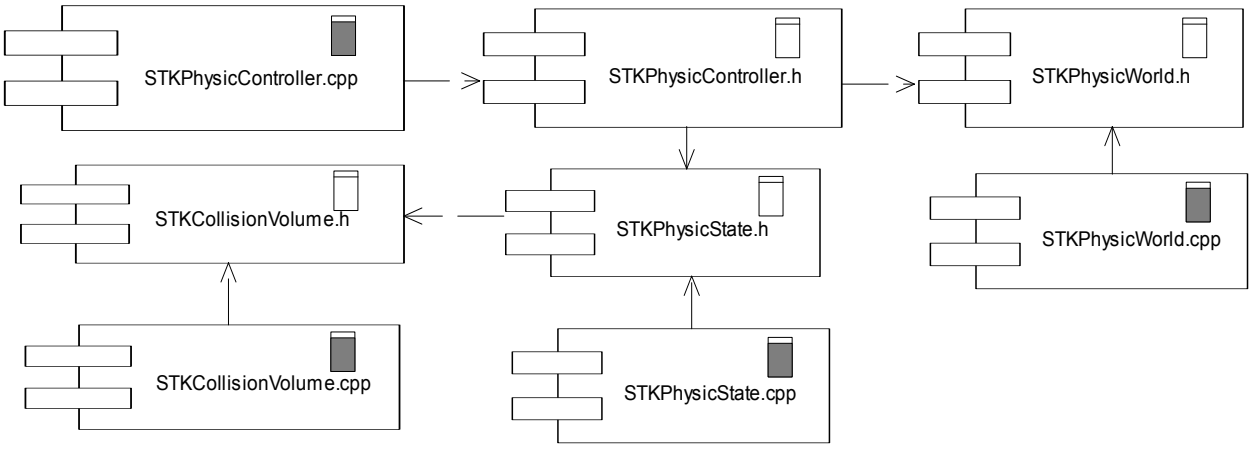


Figura 46 Diagrama de Componentes para la Actualización del estado del CR

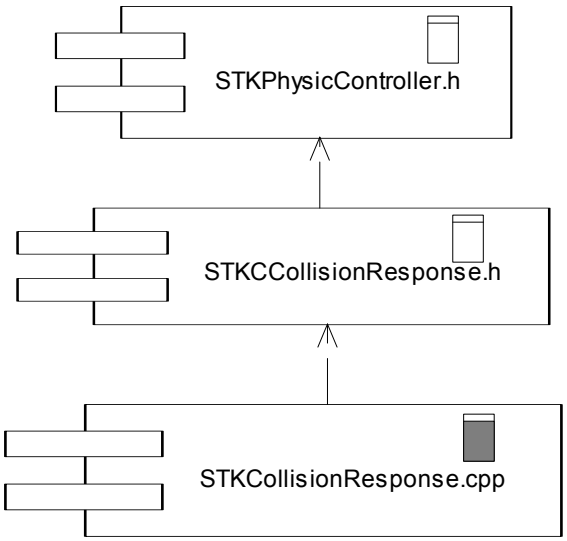


Figura 47 Diagrama de Componentes: Responder a Colisión

Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del módulo y la secuenciación de pasos traducida a mensajes entre clases de los casos de usos a desarrollar. Después de realizar la distribución de las clases en los distintos componentes se encuentra todo listo para pasar a la etapa de programación de los casos de uso. Como posibilidad adicional que brinda la herramienta Rational Rose, ya es posible generar el código fuente de los componentes.

Conclusiones Generales

Para el cumplimiento del objetivo propuesto en este proyecto se realizó un estudio del comportamiento físico matemático de los cuerpos rígidos y las leyes generales que los afectan. Se investigó como lograr con técnicas óptimas la simulación eficiente basada en la física en los SRV en tiempo real. Y a partir de la investigación realizada se propone las características técnicas de la solución propuesta.

Posteriormente se avanza por los distintos flujos de trabajo que proporciona la metodología RUP (requisitos, análisis, diseño e implementación) para lograr un producto final de calidad y que proporcione una solución al las necesidades del cliente.

Se implementó un algoritmo para la simulación del comportamiento dinámico de los cuerpos rígidos en un entorno virtual. Para crear el realismo en el mundo físico se utilizaron: la ley de la conservación del momento en la respuesta a la colisión, las fuerzas de fricción entre superficies, la fuerza de fricción de resistencia del medio, la fuerza de gravedad y la fuerza de Arquímides.

Con el módulo desarrollado se puede afirmar que se simula, con aceptable aproximación, el comportamiento dinámico de los cuerpos rígidos teniendo en cuenta sus propiedades, bajo la acción de las leyes físicas implementadas. Su incorporación a la herramienta “Scene ToolKit” servirá para lograr un mayor realismo en los proyectos finales que se elaboren.

Recomendaciones

Se recomiendan incorporar al módulo creado la simulación de cuerpos compuestos (con partes móviles y/o articulaciones). Simular otros tipos de interacciones como pueden ser las eléctricas y las magnéticas. Permitir además simulaciones precisas del comportamiento de los cuerpos en un flujo aéreo e hidrodinámico.

Referencias bibliográficas

1. LEON, I., *Ambientes de aprendizaje hipermediales: 8. Realidad Virtual* 2006.
2. GUBERN, R., *Del bisonte a la realidad virtual: V. La escena laberíntica: la realidad virtual*. 2006.
3. Luis Palomino Ramírez, I.R.G., *Diagonalización del Tensor de Inercia para mejorar la Eficiencia en la Simulación de Cuerpos Rígidos*. 1997.
4. Parejo, J.C., *Introducción a la Animación Asistida por Ordenador*. 2004.
5. Parejo, J.C., *Especificación y Control del Movimiento*. 2004.
6. Informática., D.d., *Ampliación de la Informática Grafica Tema 5 Animación 3D* 2007, Universidad de Valencia.
7. Young-Min Kang, H.-G.C., *Real-time Animation of Complex Virtual Cloth with Physical.Plausibility and Numerical Stability.* Teleoperators & Virtual Environments, 2004.
8. Lifshitz., L., *Mecánica: Mecánica del sólido rígido.* . Reverté ed. 1991, Barcelona.
9. *Rigid Tutorial*. [cited; Available from: http://www.cs.unc.edu/~ehmann/RigidTutorial.
10. Rosario, L.M., *Deformación Plástica. Mecánica de sólidos deformables.* 2004.
11. Barreiro, L.N.P., *Taller Multimedia I.Capítulo 9: Animación*, U.M. México, Editor. 2004.
12. Alba, R.V., *Dinámica del Cuerpo Rígido. Capitulo 5: Cinética de los cuerpos rígidos en el plano.*, I.T.d. Querétaro, Editor. 2006.
13. Parejo, J.C., *Animación Físicamente Basada*, U.d.S.D.d.M.A. 1., Editor. 2004.
14. AndrewWitkinz, J.P.c.S.M.S.M.E.Z.P.c., *Interactive Manipulation of Rigid Body Simulations*.
15. Ali Arafat Lemus Monterroso, J.R.A.P., *Algoritmo de Búsqueda de Ruta en un Game Engine*. 2002, Universidad Francisco Marroquin FACULTAD DE INGENIERÍA DE SISTEMAS, INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN GUATEMALA.
16. Curello, P.A., *Simulación de la Dinámica de los Cuerpos Rígidos en Tiempo Real*: Instituto Tecnológico de Buenos Aires.
17. Microsoft. *MSDN Library. Quaternion Structure.* 2007 [cited; Available from: http://msdn2.microsoft.com/en-us/library/microsoft.windowsmobile.directx.quaternion.aspx
18. Baraff, D., *Physically Based Modeling Rigid Body Simulation*, P.A. Studios, Editor.
19. CHRISTOPHER, T., *Mathematics for game developers*
20. (NASA)., N.A.a.S.A., *Newton's Law of Motion*, T. Benson, Editor. 2006.
21. Chinea, C.S. *Una Incursión por el Sólido Rígido. El tensor de inercia diagonalizado. Ejes y momentos principales* **Volume**,
22. Fiedler, G., *Game Physics: Integration Basics.* Gaffer on Games, 2006.
23. *Capitulo II. Soluciones Numéricas de Ecuaciones Diferenciales*. 2004, Buenos Aires: Universidad Nacional del Comahue.
24. Ruiz, D.G., *Motores de Física.* 2005.
25. *List of games using physics engines.*
26. Larman, C., *UML y Patrones. Introducción al análisis y diseño orientado a objetos* Editorial Félix Varela ed. 2004, La Habana.

Bibliografía consultada

1. Stahler Wendy **.** *Beginning Math and Physics for Game Programmers*. Editor, New Riders Publishing. March 24, 2004.
2. Tremblay Christopher. Mathematics for Game Developers. 2004.
3. Cortés Parejo José. *Introducción a la Animación Asistida por Ordenador*.2004
4. Hecker Chris. *Physics, The Next Frontier*. 1997
5. Hecker Chris. *Physics, Part 2 Angular Effects*. 1997
6. Hecker Chris. *Physics, Part 3 Collision Response*. 1997
7. Hecker Chris. *Physics, Part 4 The Third Dimention*. 1997
8. Hahn .James K. *Realistic Animation of Rigid Bodies*. 1988
9. Back Matthias, Schomer Elmar. *Interactive Rigid Body Manipulation with Obstacle Contacts*. 2001
10. Egan Kevin.*Techniques for Real-Time Rigid Body Simulation*. 2003. Providence, Rhode Island
11. Baraff David. *Physically Based Modeling Rigid Body Simulation*. 2001
12. Ladislav Kavan. *Rigid Body Collision Response*. 2003
13. Baraff David. Analytical *Methods for Dynamic Simulation of Non-penetrating Rigid Bodies*. Julio.1989
14. Baraff David. *Coping with Friction for Non-penetrating Rigid Body Simulation*. 1991
15. Baraff David. *Fast Contact Force Computation forNonpenetrating Rigid Bodies*. 1994
16. Witkin Andrew. *Physically Based Modeling.Course Organizar*.Pixar Animation Studios.2001
17. Erleben Kenny. *Physics Primer Basic Rigid Body Physics*. 2002

Apéndices

Glosario de abreviaturas

ANSI: American National Standards Institute (Instituto Nacional Estadounidense de Estándares): organización sin ánimo de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas.

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

CR: Cuerpo Rígido.

RV: Realidad Virtual.

SDK: Software Development Kit(kit de desarrollo de software): es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto.

SRT: Escalar-rotar-trasladar.

SRV: Sistemas de Realidad Virtual.

UCI: Universidad de la Ciencias Informáticas

3D: Tres dimensiones, en la mayoría de los casos se representa con las letras x,y,z.

Glosario de términos

Interactivo: que permite una interacción, a modo de diálogo, entre un ordenador y el usuario

Espacio tridimensional: Espacio físico-matemático que posee 3 dimensiones, comúnmente llamadas: largo, ancho y profundidad.

Leyes físicas: es una ley científica que constituye una proposición física confirmada que afirma una relación constante entre dos o más variables físicas, cada una de las cuales representa (al menos parcial e indirectamente) una propiedad de sistemas físicos concretos. Se define también como una regla y norma constante e invariable de las cosas, nacida de la causa primera o de las cualidades y condiciones de las mismas.

Inercia cinética: propiedad de los cuerpos de oponerse a un cambio de su estado de movimiento.

Transformación: Acción y resultado de transformar y transformarse. Cambiar una cosa en otra. Hacer cambiar de forma o aspecto.

Simulación: es la ejecución (habitualmente computerizada) de un modelo que reproduce el comportamiento de un sistema sometido a unas condiciones predeterminadas, posiblemente cambiantes en el tiempo.

Módulo: es un componente auto controlado de un sistema, el cual posee una interfaz bien definida hacia otros componentes; algo es modular si es construido de manera tal que se facilite su ensamblaje, acomodamiento flexible y reparación de sus componentes.

Comportamiento *(behavior):* cambio de atributos o características de los objetos visibles y no visibles del mundo virtual y que cambian el aspecto visual de la escena logrando una animación.

Dinámica: es la rama de la mecánica que estudia las causas que hacen cambiar un movimiento.

Entorno: lo que rodea a alguien o a algo. Condiciones o circunstancias físicas, humanas, sociales, culturales, etc., que rodean a las personas, animales o cosas.

Objeto simétrico homogéneo: Cuerpo físico que posee algún tipo de simetría y que su masa se encuentra homogéneamente distribuida.

Objeto irregular: Objeto cuya forma espacial carece de regularidad geométrica.

Ciclos del reloj: También denominado cielos por segundo o frecuencia, este término hace referencia a la velocidad del procesador incorporado en la CPU del ordenador, y se mide en megaherzios (MHz). A mayor índice de frecuencia, más rápido es el procesador y, en consecuencia, el ordenador.

Matriz simétrica: es una matriz cuadrada en la que los elementos simétricos respecto de la diagonal principal son iguales, esto es cuando $a_{ij} = a_{ji}$.

Sólido homogéneo: Cuerpo sólido que su masa se encuentra homogéneamente distribuida.

Animación: simulación de un movimiento creada por la muestra de una serie de imágenes o fotogramas.

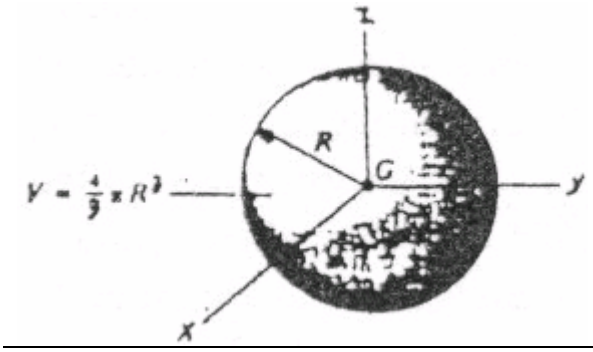
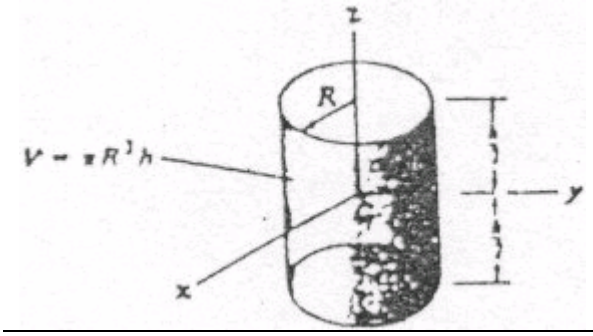
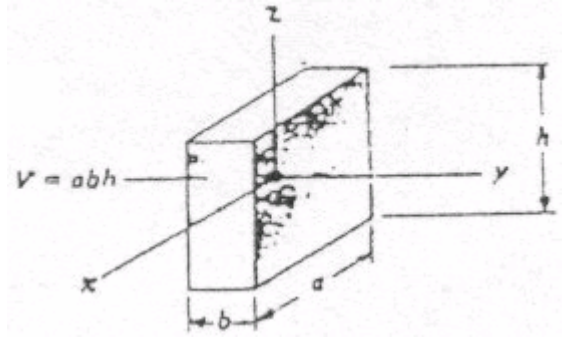
Modelo físico: La física intenta entender el mundo haciendo modelos de la realidad, usados para racionalizar, explicar y predecir fenómenos físicos a través de una teoría. Hay tres tipos de teorías físicas: teorías aceptadas, teorías propuestas y teorías no aceptadas. Algunas teorías físicas son desechadas por la observación mientras que otras no. Una teoría física es un modelo de eventos físicos y no puede ser probado por axiomas básicos. Una teoría física es diferente a un teorema matemático. Los modelos de teorías físicas son la realidad y una declaración de lo que se observa así como la predicción de nuevas observaciones.

Motores físicos: simulan la dinámica de los cuerpos rígidos en los juegos y simuladores actuales.

Tiempo real: es la capacidad de procesar una muestra de señal antes de que ingrese al sistema la siguiente muestra.

Escena: Cada una de las partes de que consta una obra dramática, una película, o una animación y que representa una determinada situación, con los mismos personajes u objetos.

Tensores de Inercia

Sólido Homogéneo	Matriz del Tensor de Inercia
Esfera 	$\begin{pmatrix} \frac{2}{5}MR^2 & 0 & 0 \\ 0 & \frac{2}{5}MR^2 & 0 \\ 0 & 0 & \frac{2}{5}MR^2 \end{pmatrix}$
Cilindro 	$\begin{pmatrix} \frac{1}{12}M\left(3R^2+h^2\right) & 0 & 0 \\ 0 & \frac{1}{12}M\left(3R^2+h^2\right) & 0 \\ 0 & 0 & \frac{1}{2}MR^2 \end{pmatrix}$
Bloque rectangular 	$\begin{pmatrix} \frac{1}{12}M\left(a^2+h^2\right) & 0 & 0 \\ 0 & \frac{1}{12}M\left(b^2+h^2\right) & 0 \\ 0 & 0 & \frac{1}{12}M\left(a^2+b^2\right) \end{pmatrix}$

Índice de figuras y tablas

Índice de figuras

Índice de tablas

Estándares de codificación

El código de la biblioteca sigue algunos estándares propuestos por el grupo de desarrollo (respetando los estándares de codificación para C++ (uso de espacios y líneas en blanco, etc.)). Está programado en inglés, debido que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático.

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

Nombre de los ficheros:

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

Constantes:

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

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

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

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

enum **E**Node**T**ype {**NT**_GEOMETRYNODE,…};

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

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

Clases: class **C**ClassName;

Indicando con "**C**" que es una clase

Interfaces: **I**MyInterface

Indicando con "**I**" que es una interfaz.

Listas e iteradotes STD:

vector<> **T**Name**List**;
TNameList::iterator **T**Name**ListIter**;

map<> **T**Name**Map**;
TNameMap::iterator **T**Name**MapIter**;

multimap<> **T**Name**MultiMap**;
TNameMultiMap::iterator **T**Name**MultiMapIter**;

Declaración de variables:

Los nombres de las variables comenzarán con un identificado 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 antepondrá el identificador "" (en minúscula), si son globales se les antepondrá la letra "g", y en caso de ser argumentos de algún método, se les antepondrá el prefijo "arg_".

Tipos simples:
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
char **g***c*GlobalVarName; //variable global, no se le antepone “”
short **s**Name;

Instancias de tipos creados:
EMyEnumerated **e**Name;
SMyStructure **k**Name;
CClassName **k**ObjectName;
CClassName* **pk**Name; //puntero a objeto
CClassName* **ak**Name; //arreglo de objetos
CClassName* **ak**Name; // variable miembro de clase
IMyInterface* **pI**Name; //puntero interfaces

Métodos

En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada. Solamente los constructores y destructores comenzarán con “”.

En el caso de los argumentos se les antepone el prefijo “arg_”

Constructor y destructor:

CClassName (bool **arg_**bVarName, float& **arg_**fVarName);
~**C**ClassName ();

Funciones:
b*ool bFunction1 (…);*
int* **pi**Function2 (…);
CClassName* **pk**Function3 (…);

Procedimientos:
void Procedure4 (…);

Métodos de acceso a miembros

Los métodos de acceso a los miembros de las clases no se nombrarán “Gets” y “Sets”, sino como los demás métodos, pero **con el nombre** de la variable a la que se accede y sin “m_”:

int **i**MyVar; //variable

Obtención del valor:

```
int iMyVar();
{
   return iMyVar;
}
```

Establecimiento del valor:

```
void MyVar(char* arg_iMyVar)
{
  iMyVar = arg_iMyVar;
}
```

Obtención y establecimiento del valor:

```
int& iMyVar();
{
   return iMyVar;
}
```