Universidad de las Ciencias Informáticas
Facultad 5

Título: Integración de la dinámica de los cuerpos rígidos a los Laboratorios Virtuales.

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

Autor:
Javier Alejandro Domínguez Falcón

Tutora:
Msc. Liudmila Pupo Peña
Co-Tutor:
Ing. Alejandro David Merzeau

La Habana 2014
”Año 56 de la Revolución**”**

"Tu tiempo es limitado, de modo que no lo malgastes viviendo la vida de alguien distinto. No quedes atrapado en el dogma, que es vivir como otros piensan que deberías vivir. No dejes que los ruidos de las opiniones de los demás acallen tu propia voz interior. Y, lo que es más importante, ten el coraje para hacer lo que te dicen tu corazón y tu intuición."

Steve Jobs

Declaración de autoría

Declaro ser autor del presente trabajo “Integración de la dinámica de los cuerpos-rígidos a los Laboratorios Virtuales” y reconozco a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmo la presente a los ___ días del mes de _____ del año 2014.

_____

Javier Alejandro Domínguez Falcón

Autor

_____ _____

Msc. Liudmila Pupo Peña

Tutor

Ing. Alejandro David Merzeau

Co-Tutor

Datos de contacto

Tutor: Msc. Liudmila Pupo Peña.
Especialidad de graduación: Ingeniería en Ciencias Informáticas
Categoría docente: Asistente
Categoría científica: Máster en Informática Aplicada
Años de experiencia en el tema: 10
Año de graduado: 2007
Correo electrónico: lpupo@uci.cu

Co-Tutor: Ing. Alejandro David Merzeau
Especialidad de graduación: Ingeniería en Ciencias Informáticas
Categoría docente: -
Categoría científica: -
Años de experiencia en el tema: 5
Año de graduado: 2012
Correo electrónico: admerzeau@uci.cu

Siempre resultará difícil (aunque agradable por cierto) agradecer a todos aquellos que de una u otra forma me han acompañado, porque no alcanza el tiempo o la memoria para dar con justicia los créditos y méritos a quienes se lo merecen. Diciendo de antemano MUCHAS GRACIAS, quiero agradecer:

A mis padres, que son lo más grande que tengo en la vida y a quienes debo todo lo que soy. Gracias por estar a mi lado en todos los momentos. A mi hermana, a quien quiero mucho. A toda mi familia, a mis abuelos, tios, primos…, que de una u otra forma han estado siempre presentes.

A mi linda novia que se ha mantenido junto a mí y me ha brindado todo su amor y cariño.

A mi hermano y sobrino que aunque hemos pasado la mayor parte del tiempo separados los quiero mucho.

A mis primos Melanie, Randy, Ale, Luisito, Otto, a mis tías Adiuska, Ricardo, Caridad, Lidia, Norma, Isabel, gracias por complementar mi felicidad con su existencia.

A mis abuelos por la ternura con la que me han tratado, a mi abuela MIMI que sin ella no sabría que fuese de mi vida. Todos ustedes ocupan un lugar muy especial dentro de mí.

A mis suegros que son para mí como un padre y una madre, los dos han sido conmigo incondicionales. Son dos personas que admiro y quiero mucho. Gracias por toda su ayuda.

A todos mi compañeros de carrera, demasiados para que los mencione a todos, pero no para que los recuerde a todos. Si seguí adelante fue gracias a los ánimos que me dieron. Si seguí mejorando fue gracias a vuestras críticas. De no ser por ustedes, no me habría graduado…

A los del apartamento de Lianet: Elizabeth, Loreni, Isenic, Alejandro , Keimer. Agradecerle todos los detalles que tuvieron conmigo los frijoles que me hacían a diario. Para mí son como una familia.

A dos personas que estuvieron presentes cuando de veras los necesité, para apoyarme, darme ánimo, y decirme que si lo lograría, Alejandro y Carlos, sin ustedes hubiera sido imposible.

A Alejandro que más que un amigo para mí es un hermano…

A mis tutores, Liudmila y Alejandro David, por haberme brindado sus consejos y ayuda, aprendí mucho de ustedes y a mi tercer tutor Yandy por dedicarme tiempo y ayuda sin decir nunca que no.

Dedico este trabajo a mis cuatro lindos abuelos, aunque dos de ellos ya no estén conmigo. Mi abuelo PIPO aunque se haya perdido gran parte de mi vida siempre lo llevaré en mi corazón y sé que estaría muy orgulloso de mi.

Mi abuela MARIANA fue quien nunca dejó de decirme que mis estudios, en esta etapa, eran lo más importante. Gracias abuela por todos los consejos que mediste y por todo lo que me enseñaste. Este triunfo es también tuyo.

A mi mamá y mi papá, que me han enseñado a ver la realidad con mis propios ojos, que sienten mis logros como si fueran suyos, que ven en mis sueños los suyos; a ustedes que admiro y respeto los amo con todo mi corazón. Gracias a los dos por demostrarme que cambiarían todo por ofrecerme un instante de felicidad.

A mi hermana, por su gran amor y cariño, por siempre estar juntos en los bueno y malos momentos; por siempre aceptar todos mis consejos y regaños sin ningún problema.

Quizás no sean palabras tan bonitas, pero serán las más sinceras que te puedo decir, no quiero divagar buscando decir cosas que suenen demasiado grandes para este momento, pero no sabes cuanto agradezco haberte conocido, y sin dudas has sido una parte fundamental en mi formación como profesional y como persona. Gracias por estar a mi lado todo este tiempo apoyándome, cuidándome y escuchándome. Gracias por ser mi Compañera, Amiga y Novia. Gracias mi PITI BUUU.

Resumen

Las aplicaciones de Realidad Virtual en los últimos años han tenido gran auge en un amplio número de sectores de la sociedad. Los Laboratorios Virtuales son un ejemplo de estas aplicaciones que permiten experimentar y en cierta medida, poder palpar el resultado de un desenvolvimiento y actividad dentro de un ambiente tridimensional creado artificialmente.

El realismo de las escenas simuladas en estas aplicaciones 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, lo que provoca que disminuya el realismo de las escenas y por tanto la percepción de inmersión en un espacio tridimensional. En el Centro de Entornos Interactivos 3D de la Universidad de Ciencias Informáticas se desarrollan Laboratorios Virtuales sobre una arquitectura basada en capas y componentes. El objetivo de este trabajo es implementar un componente dinámico que permita integrar la dinámica de cuerpos rígidos a la arquitectura de los Laboratorios Virtuales.

En la investigación realizada se presenta un componente dinámico que hace uso de una biblioteca para realizar los cálculos físicos nombrada capa de abstracción física. Esta capa permite incorporar física a las aplicaciones con un bajo costo de desarrollo, además posibilita el uso de múltiples motores físicos en una misma aplicación.

El componente resultante de este trabajo, está desarrollado usando el proceso unificado de software e implementado en c++ estándar. Permite un mayor realismo en los productos finales que se elaboren en los Laboratorios Virtuales y logra incorporar las leyes físicas que determinan el comportamiento de los cuerpos rígidos, según factores externos (como fuerzas) que incidan sobre él.

Palabras clave: Capa de Abstracción Física, Componente Dinámico, Cuerpos Rígidos, Laboratorios Virtuales.

Índice

Índice de figuras

Índice de ecuaciones

Índice de tabla

Índice de Gráfica

Introducción

En la actualidad los avances de las Tecnologías de la Información y las Comunicaciones (TIC) han permitido intervenir en un gran número de procesos en diversas esferas de la sociedad, provocando un impacto en todos los ámbitos de la vida. En el campo de la educación, las TIC se han insertado como una herramienta que aún tienen mucho que aportar; pueden contribuir al acceso universal a la educación, al desarrollo profesional de los docentes, suministrar medios para la mejora de los procesos de enseñanza y aprendizaje, así como lograr una gestión más eficiente del sistema educativo.

Los espacios de Realidad Virtual (RV) han llegado a adquirir un gran auge no solo en la Informática, sino también en la vida diaria de las personas. Son muchas las aplicaciones de este tipo que se utilizan en disímiles campos, así como: en la medicina (quirófano virtual), en la química (diseño de compuestos químicos), en el entretenimiento (juegos) y en el ejército (los simuladores de vuelo, herramienta para el entrenamiento de pilotos). También se benefician todas las profesiones que utilizan habitualmente maquetas o prototipos, pues la RV permite al diseñador presentar a sus clientes simulaciones en tres dimensiones 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 el entrenamiento de personal especializado sin poner en peligro su integridad física.

Los Laboratorios Virtuales (LV) son herramientas que permiten experimentar y en cierta medida, palpar el resultado de un desenvolvimiento y actividad dentro de un ambiente tridimensional, creado artificialmente. Mediante esta representación se pretende aproximar al ambiente de un laboratorio tradicional, donde su objetivo fundamental es que el mundo virtual se aproxime lo mejor posible al mundo real. En estos contextos virtuales no se necesitan de los materiales y herramientas reales para la realización de las prácticas, por lo que constituyen una opción viable ante la falta de recursos.

Cuando se desarrollan aplicaciones de RV, uno de los principales desafíos es lograr aumentar el realismo e inmersión a un costo computacional aceptable. Con las placas gráficas actuales es posible obtener imágenes muy cercanas a la realidad aunque también se debe tener en cuenta la combinación de sonido, dispositivos para interactuar con el entorno, animaciones y física para lograr escenas que brinden una verdadera inmersión.

Dentro de los principales requisitos que debe tener un sistema de simulación se encuentra el hecho de que los objetos presentes en el entorno se comporten lo más parecido a la realidad al ser sometidos a fuerzas, al efectuarse un movimiento que provoque cambios en su dirección y sentido, un aumento de velocidad o variabilidad en algún punto de referencia. Esto es un proceso que incluye la implementación y simulación de las leyes físicas en los entornos virtuales lo cual resulta complejo hacerlo desde cero y requiere gran cantidad de tiempo. Sin embargo, esta tarea puede verse reducida a una configuración de los objetos de la escena y de sus propiedades, si se cuenta con otra aplicación para recrear la simulación deseada.

Actualmente, se puede encontrar en los mercados diferentes opciones de plataformas de software tanto basados en licencias libres como comerciales, llamadas motores físicos, encargadas de simular el comportamiento de un conjunto de objetos en un mundo tridimensional. La variedad y cantidad de motores físicos disponibles en el mercado amerita un arduo trabajo para los desarrolladores al momento de decidir o portar sus aplicaciones de un motor a otro, pues cada uno posee su propio conjunto de características, métodos de inicialización e interfaces (1).

La Universidad de las Ciencias Informáticas (UCI) cuenta con el Centro de Entornos Interactivos 3D (VERTEX) que posee un proyecto que desarrolla LV de distintas materias como ensamblado de computadoras, electrónica analógica básica, taller eléctrico, instrumentación y control. El desarrollo de estos LV se realiza sobre una arquitectura con estilos basados en capas y en componentes. Esta arquitectura brinda como principal funcionalidad la posibilidad de convertir los procesos más comunes a todos los LV en componentes, lo que permite realizar optimizaciones y refinamientos, proporcionando un amplio nivel de reutilización. Es posible ensamblar un cierto número de componentes y lograr la funcionalidad básica de un LV, reduciendo los costos de desarrollo.

Los productos que se desarrollan sobre esta arquitectura tienen la necesidad de otorgar mayor realismo a las escenas simuladas que no siempre tienen la calidad requerida. Uno de los principales problemas es que no se presentan las leyes físicas elementales que rigen el comportamiento de los cuerpos, lo que provoca que disminuya el realismo de las escenas y por tanto la percepción de inmersión en un espacio tridimensional.

Por la situación anteriormente descrita se identifica el siguiente **problema de la investigación**: ¿Cómo lograr la incorporación de la dinámica de los cuerpos rígidos a los entornos virtuales que se desarrollan con la arquitectura para Laboratorios

Virtuales desarrollada en el centro VERTEX de la Universidad de las Ciencias Informáticas?

El **objeto de estudio** de este trabajo lo constituye la dinámica de los cuerpos rígidos, enmarcado en el **campo de acción** la incorporación de la dinámica de los cuerpos rígidos en los entornos virtuales.

Para dar solución al problema de la investigación se ha definido como **objetivo general**: Implementar un componente dinámico que permita integrar la dinámica de cuerpos rígidos a la arquitectura de LV.

Para cumplir el objetivo planteado se proponen las siguientes **tareas de investigación:**

- Levantamiento de los requisitos para identificar los principales elementos físicos a incorporar en los LV.
- Investigación de las leyes de la dinámica de cuerpos rígidos para comprender las características y funcionalidades de los motores de simulación física existentes.
- Análisis de la arquitectura de los LV para poder integrar un componente dinámico que permita la simulación de dinámica de los cuerpos rígidos en los LV.
- Comparación de los motores de simulación física existentes para seleccionar el motor de simulación física que mejor se adapte a los requisitos definidos.
- Diseño e implementación del componente dinámico para los LV siguiendo las especificaciones de la arquitectura.
- Obtención de un demo que muestre la integración del motor de simulación física a la arquitectura de los LV.

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

En un primer capítulo, “Fundamentos Teóricos”, se hace un análisis bibliográfico donde se investigan las leyes físico-matemáticas elementales que rigen el comportamiento de los objetos y las principales características de los motores de simulación física. En el capítulo 2, “Descripción de la solución técnica”, se selecciona el ambiente de desarrollo para el componente de la física, se realiza una comparación de los motores de simulación física y se selecciona el que mejor se adapte a las necesidades del proyecto, se analiza con mayor profundidad el objeto de estudio, se crea el modelo del dominio, se plantean las reglas del negocio, se hace la captura de requisitos y se crean los modelos de casos de uso del sistema. En el capítulo 3,

“Diseño del sistema”, se presentan los patrones de diseños, los diagramas de clases. En el capítulo 4, “Implementación y prueba”, en este capítulo se hace referencia a la codificación y pruebas del sistema. Finalmente, se ofrece un glosario de términos para ayudar a la comprensión del lenguaje técnico utilizado a lo largo del trabajo.

Capítulo 1: Fundamentos teóricos

Introducción

En la literatura se pueden encontrar muchas definiciones de simulación, una de las más completas es la que propone Robert Shannon:*"La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias - dentro de los límites impuestos por un cierto criterio o un conjunto de ellos - para el funcionamiento del sistema"* (2).

Para lograr una simulación lo más real posible se necesita producir la impresión de integración física en un ambiente tridimensional que logre cumplir con todos los requisitos que el sistema necesite. Es necesario que los objetos en el mundo sean dinámicos y cumplan con las leyes físicas elementales que rigen el comportamiento de los cuerpos. Esto exige seleccionar de forma minuciosa un motor de simulación física el cual permite crear escenas con una verdadera inmersión y realismo.

En este capítulo se realiza un estudio de las leyes físicas elementales y se definen las características de la simulación dinámica de cuerpos rígidos. Se analizan las principales características de los motores de simulación física.

1.1 Simulación interactiva

De los diferentes órganos de los sentidos, la vista es la que proporciona una mayor cantidad de información y a la vez es la que da una mayor sensación de presencia. Por este motivo, todo sistema de RV debe proporcionar estímulos adecuados como mínimo para el sentido de la vista (es decir, debe generarlas imágenes lo suficientemente realistas).

Es necesario mencionar la distinción entre simulación gráfica y RV. La simulación gráfica no implica la inmersión en un entorno a través de los sentidos sino que se encarga de llevar a la computadora una representación del mundo real lo cual no implica interacción. Por el contrario, la RV es una realidad simulada por computador capaz de interactuar con todos los sentidos o al menos con los mínimos necesarios para producir en el cerebro una sensación de inmersión en el entorno simulado (3).

El hecho de que la simulación sea interactiva es lo que distingue la RV de una animación. En una animación, al igual que en el cine, los espectadores son individuos pasivos en el sentido que no pueden alterar el contenido de las imágenes que ven. En cambio, en un sistema de RV, el usuario puede escoger libremente su movimiento por la escena y por tanto, sus acciones afectan de forma directa a las imágenes que verá.

El sistema de RV responderá en tiempo real (es decir, con un tiempo de respuesta despreciable) a sus acciones (4).

Si se quiere lograr realismo es necesario tener buena calidad y credibilidad del movimiento de los objetos. Aunque teóricamente la animación puede ser producida manualmente por animadores, el trabajo se dificulta cuando la complejidad de la escena crece. La alternativa natural es usar la física para modelar el movimiento de los objetos: esta opción provee realismo con un mínimo esfuerzo utilizando un software capaz de simular animaciones basadas en la física (5).

1.2 Dinámica del cuerpo rígido

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.

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 (6).

La mecánica puede definirse como la ciencia que describe y predice las condiciones de reposo o movimiento de los cuerpos bajo la acción de fuerzas. Se divide en tres partes: mecánica de los cuerpos rígidos, mecánica de los cuerpos deformables y mecánica de fluidos.

La mecánica de los cuerpos rígidos se subdivide en estática y dinámica, refiriéndose la primera a los cuerpos en reposo y la segunda a los cuerpos en movimiento. En esta parte del estudio de la mecánica se supone que los cuerpos son perfectamente rígidos. Las estructuras y máquinas reales, sin embargo, no son nunca absolutamente rígidas y se deforman por la acción de las cargas a las que están sometidas. Pero estas deformaciones son pequeñas normalmente y no afectan de manera apreciable las condiciones de equilibrio o movimiento de la estructura considerada (7).

A continuación se muestran algunas definiciones que permiten una correcta interpretación del término cuerpo rígido.

Un cuerpo rígido (CR) o sólido rígido es un supuesto cuerpo que no se deforma, aunque sin embargo nunca son absolutamente rígidos pues todos se deforman bajo las cargas a las que están sometidas (7).

Los cuerpos reales llegan a ser muy complejos, las fuerzas que actúan sobre ellos pueden deformarlos, estirarlos, torcerlos y aplastarlos. Al ignorar tales deformaciones y suponer que el cuerpo tiene forma y tamaño perfectamente definido e inmutable, llamamos a este modelo idealizado CR (8).

La forma y comportamiento de los cuerpos quedan definidos al especificar sus propiedades físicas, tales como dimensiones, masa, fuerzas y gravedad. El movimiento de un CR en el espacio 3D depende únicamente de tres aspectos: las fuerzas externas aplicadas al cuerpo, la masa y la forma del cuerpo, originando una posición, velocidad y aceleración del cuerpo en el tiempo, tanto de translación como de rotación (9).

Los cuerpos **no rígidos** o **deformables**, son aquellos que sufren una deformación (cambio del tamaño o la forma) debido a la aplicación de una o más fuerzas sobre él (gravedad, viento, pinza, mano) (10).

Para realizar simulaciones realistas, hay que tener en cuenta las leyes físicas, que utiliza la cinemática y la dinámica (11).

La cinemática de cuerpos rígidos, es la parte de la mecánica que describe el movimiento (8). Aquí se estudia el movimiento de los cuerpos, sin entrar a considerar la causa que lo originó.

La cinemática del sólido rígido estudia el movimiento de los cuerpos cuando se consideran sus rotaciones (12).

La dinámica, es el estudio del movimiento de los cuerpos bajo la acción de las fuerzas que actúan sobre ellos (12).

Utilizando la dinámica de los CR es posible obtener gran realismo, pero resulta difícil especificar la animación. Hay que tomar en consideración masas, aceleraciones, restricciones al movimiento. La dinámica de los CR 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).

1.3 Conceptos matemáticos

1.3.1 Transformaciones Geométricas

En la representación de gráficos 3D es necesario contar con herramientas para la transformación de los objetos básicos que compondrán la escena. Los objetos usualmente están representados por conjuntos de triángulos que definen mallas poligonales. Las operaciones que se aplican a estos triángulos para cambiar su posición, orientación y tamaño se denominan transformaciones geométricas (13).

1.3.2 Sistema de Coordenadas

Posición

En un mundo virtual se necesitan posicionar objetos en el espacio 3D. El motor gráfico debe gestionar por tanto la posición, orientación y escala de estos objetos (y sus cambios a lo largo del tiempo).

Para representar la posición de una partícula en el espacio es necesario establecer un sistema de coordenadas. Este consiste en un punto de origen X cuyos tres componentes son (Xx, Xy, Xz) y tres vectores que determinan su orientación: Vx, Vy, Vz como se muestra en la Figura 1 (14).

Un punto puede definirse como una localización en un espacio n-dimensional, este espacio suele ser bidimensional o tridimensional.

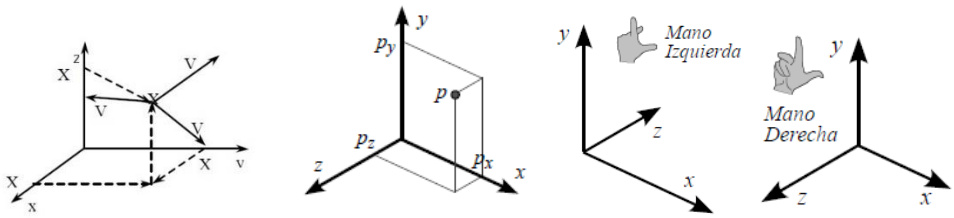


Figura 1. Coordenadas cartesianas

Coordenadas Cartesianas. Es el sistema de coordenadas más habitual. Este sistema de coordenadas define ejes perpendiculares para especificar la posición del punto en el espacio. Como se muestra en la Figura 1, existen dos convenios para la definición de los ejes de coordenadas cartesianas; según la regla de la mano derecha o de la mano izquierda. Es muy sencillo convertir entre ambos convenios; basta con invertir el valor del eje que cambia (13).

Si se desea describir, medir y analizar las características geométricas de los objetos hay que colocarlos dentro de un sistema de coordenadas. El estudio se centra en dos sistemas:

Sistema de Coordenadas Global o del mundo

Sistema de Coordenadas Local

Las primeras se toman respecto de un sistema de coordenadas principal que se encuentra en el punto (0,0,0) y cuyos ejes son Vx =(1,0,0), Vy=(0,1,0), Vz=(0,0,1) en principio todo vector está representado en estas coordenadas (14).

Las coordenadas locales se toman respecto de un sistema de coordenadas que se mueve y rota junto con un cuerpo en particular. Esto es sumamente útil pues todo punto que se mantenga fijo respecto a un cuerpo puede ser expresado en coordenadas locales sin importar si el objeto está rotando o moviéndose, es decir, en coordenadas globales es un punto constante mientras que en coordenadas locales es variable (el sistema de coordenadas se mueve y rota con el objeto o cuerpo) (14).

La transformación básica bidimensional más sencilla es la traslación. Para realizar la traslación de un punto p a otro punto p´ solo se necesita sumar a las coordenadas iniciales del punto p un vector de desplazamiento y se obtendrá la nueva posición de coordenadas p´. Para trasladar un objeto en mundo tridimensional aplicamos esta traslación a todos los puntos del objeto y estaríamos desplazando ese objeto de una posición a otra. De este modo, se puede definir la traslación como la suma de un vector libre de traslación t a un punto original p para obtener el punto trasladado p′ (ver Figura 2) (13).

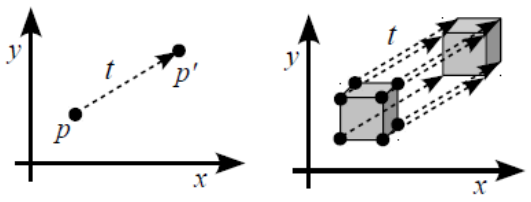


Figura 2. Sistema de coordenadas locales

En la Figura 2 se muestra la traslación de un punto p a p' empleando el vector t. Es posible trasladar un objeto poligonal completo aplicando la traslación a todos sus vértices.

Si a un CR se le aplica una fuerza, esta tiende a imprimirle tanto un movimiento de traslación como de rotación, alrededor de un eje que pasa por un punto del CR (15).

Cuando se cambia la localización de un objeto, se necesita especificar una combinación de traslaciones y rotaciones en el mismo (por ejemplo, cuando se lanza un cubo por unas escaleras a este objeto se le aplican varias traslaciones y

rotaciones). Es interesante por tanto disponer de alguna representación que permita combinar transformaciones de una forma eficiente.

1.4 Representación de rotaciones

Las formas más comunes de representar rotaciones son:

- Ángulos de Euler.

- *Quaternion*.

- Matrices.

1.4.1 Ángulos de Euler

Los ángulos de Euler 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, (ver Figura 3**)** (14).

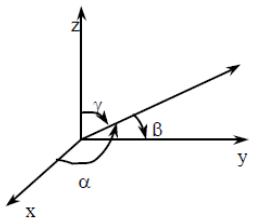


Figura 3. Ángulos de Euler

1.4.2 *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 4*x*4 usadas para las transformaciones y las rotaciones. Los *quaternions* adicionan un cuarto elemento en los valores de (*x*, *y*, *z)* que definen un vector, resultando un vector de cuatro componentes.

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 (14).

1.4.3 Representación Matricial

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.

$$\mathcal{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \mathcal{R}_y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \mathcal{R}_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figura 4. Matrices de Rotaciones

1.4.4 Matrices de rotaciones

Las matrices de transformación son muy útiles, el problema es que una matriz de 4x4 guarda 16 números y una de 3X3 guarda 9, 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 (14).

Es posible utilizar una matriz para resumir cualquier secuencia de rotaciones en el espacio 3D. Sin embargo, las matrices no son la mejor forma de representar rotaciones (13).

En el ejemplo de la Figura 5. a) se muestra el sistema de coordenadas que se utiliza, junto con las elipses de rotación asociadas a cada eje. En la Figura 5. b) se aplica una rotación respecto x, y se muestra el resultado en el objeto. El resultado de aplicar una rotación de 90º en y se muestra en la Figura 5. c) en la Figura 5. d) se muestra el resultado de aplicar una rotación sobre el eje z (13).

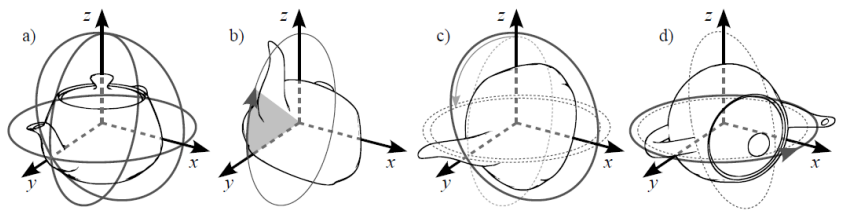


Figura 5. Representación de las transformaciones sobre los 3 ejes de coordenadas.

1.5 Magnitudes físicas

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

Una partícula es un modelo matemático, un punto, sin dimensiones pero con masa y su posición se puede especificar en el espacio (16).

Como puede observarse la partícula supone una idealización de un CR, en el que se ha despreciado su tamaño. A su vez, el CR supone otra idealización, en este caso de un medio continuo, en el que se ha supuesto que la distancia entre dos puntos del medio se mantiene constante. El medio continuo también es la idealización de un cuerpo material, tratado como un sistema discreto de moléculas separadas por espacios vacíos (16).

1.5.1 Vectores

Un **vector** es utilizado para representar una magnitud física el cual necesita de un módulo y una dirección (u orientación) para quedar definido.

Los vectores se pueden representar geométricamente como segmentos de recta dirigidos o flechas en planos $\mathbb{R}^2$ o $\mathbb{R}^3$; es decir, bidimensional o tridimensional (17).

Ejemplos

- La velocidad con que se desplaza un automóvil es una magnitud vectorial, pues no queda definida tan solo por su módulo (lo que marca el velocímetro, en el caso de un automóvil), sino que se requiere indicar la dirección hacia la que se dirige.
- La fuerza que actúa sobre un objeto es una magnitud vectorial, pues su efecto depende, además de su intensidad o módulo, de la dirección en la que opera.
- El desplazamiento de un objeto.

1.5.2 Magnitudes escalares y vectoriales

Las magnitudes físicas, tales como la masa, la presión, el volumen; quedan completamente definidas por un número y las unidades utilizadas en su medida, aparecen en otras, tales como el desplazamiento, la velocidad, la aceleración, la fuerza, que no son definidas por un dato numérico, sino que llevan asociadas una dirección. Estas últimas magnitudes son llamadas **vectoriales** en contraposición a las primeras que son llamadas **escalares** (17).

1.5.3 Velocidad lineal y angular

La velocidad lineal $\mathcal{V}(t)$ de una partícula se obtiene por medio de la siguiente derivada: $\mathcal{V}(t) = \mathcal{X}(t)'$ donde $\mathcal{X}(t)$ es la posición de una partícula que depende del tiempo.

$\mathcal{X}$ y $\mathcal{V}$ son vectores, por lo tanto $\mathcal{V}$ será un vector cuyos componentes son los de $\mathcal{X}$ derivados respecto al tiempo. Se le llama rapidez al módulo del vector velocidad, que

representa las unidades o metros por segundo que recorrería la partícula si esta rapidez se mantuviera constante (14).

La **velocidad angular** se representa por el vector $\mathcal{W}$, 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 6, donde $\mathcal{X}(t)$ es la posición del centro de masa del cuerpo, $\mathcal{V}(t)$ es la velocidad del centro de masa e $\mathcal{W}(t)$ indica el eje y rapidez de rotación del cuerpo (14).

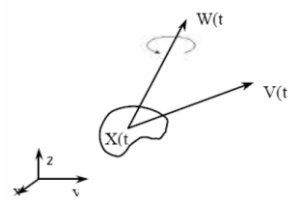


Figura 6. Velocidad lineal y angular

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 (14).

1.5.4 Propiedades de masa del cuerpo rígido

Las propiedades de masa de un sólido rígido son importantes en el estudio del comportamiento de este tipo de cuerpos, pues la velocidad (tanto lineal como angular) y la respuesta a la aplicación de una fuerza están en función de dichas propiedades. En primer lugar, la masa de un objeto es la medida que indica la resistencia que ofrece un cuerpo a ser desplazado. Cuanto mayor es la masa, más difícil será cambiar su estado de movimiento. La masa total de un cuerpo podría calcularse sumando las masas de todas las partículas elementales que lo conforman (18).

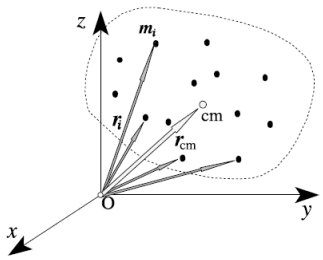


Figura 7. Centro de masa

1.5.5 Centro de masa

Se considera un sistema de partículas, compuesto por $\mathcal{N}$ de ellas, cuyas masas se designan por $m_i (i = 1, 2, ...\mathcal{N})$ y sea r_i el vector de posición de la partícula *i*-ésima respecto al origen $\mathcal{O}$ de un referencial dado. Se definió el centro de masa de un tal sistema de partículas como el punto del espacio, que se designa por cm (o por G, cuando así convenga), cuyo vector de posición respecto a $\mathcal{O}$ es

$$\boldsymbol{r_{cm} = \frac{\sum_{i=1}^{n} m_i \mathbf{r}_i}{\sum_{i=1}^{n} m_i}}$$

Ecuación 1. Centro de masa

donde m_i representa, la masa total del sistema de partículas (19).

Nota: la posición del centro de masa de un sistema es independiente del referencial que se utilice y depende solamente de las masas de las partículas y de las posiciones de unas respecto a otras (19).

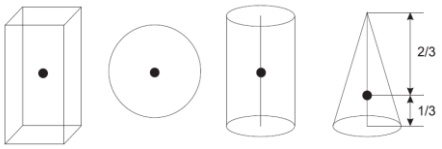


Figura 8. Centro de gravedad

1.5.6 Centro de gravedad

Se considera un cuerpo de masa $\mathcal{M}$ que se encuentra en una región del espacio donde existe un campo gravitatorio. La fuerza que actúa sobre cada una de las partículas que lo constituyen viene dada por $m_i\, g_i$, donde m_i representa la masa de la partícula *i*-ésima y g_i es la intensidad del campo gravitatorio en el punto donde se encuentra dicha partícula. La fuerza total que actúa sobre las $\mathcal{N}$ partículas que constituyen el cuerpo es la resultante general de ese sistema de fuerzas (19).

$$\boldsymbol{\mathcal{P} = \sum_{i=1}^{n} m_i g_i}$$

Ecuación 2. Fuerza gravitatoria

Si la intensidad del campo gravitatorio $\mathcal{g}$ tiene el mismo valor en todos los puntos de una cierta región del espacio, decimos que el campo gravitatorio es uniforme en dicha región. Para un cuerpo situado en un campo gravitatorio uniforme, $\mathcal{g}$ tiene el mismo valor para todas las partículas que lo constituyen, de modo que las fuerzas gravitatorias individuales forman un sistema de vectores paralelos entre sí cuya resultante es $\mathcal{P} = \mathcal{mg}$, el peso del cuerpo. El centro de ese sistema de vectores paralelos recibe el nombre de *centro de gravedad* y viene determinado por (19):

$$\mathcal{r}_{\mathcal{g}} = \frac{\sum_{i=1}^{n} \mathcal{m}_i \, \mathbf{r}_i}{\sum_{i=1}^{n} \mathcal{m}_i}$$

Ecuación 3. Centro de gravedad

1.5.7 Fuerzas y torques

Los responsables de causar cambios en la velocidad de un objeto son las fuerzas. Según la Primera Ley de Newton, todo cuerpo permanece en su estado de reposo o movimiento a menos que la acción de una fuerza le obligue a cambiarlo. En el estudio de un CR se realiza distinción entre fuerza ($\mathcal{F}$) y torque ($\boldsymbol{\tau}$). La primera es la responsable del movimiento lineal de una partícula, mientras que el segundo es el responsable del movimiento rotacional y se expresa en términos de $\mathcal{F}$. El torque se define como:

$$\boldsymbol{\tau} = (\mathcal{r} - \mathcal{x}) \times \mathcal{F}$$

Ecuación 4. Torque

donde $\mathcal{r}$ es el punto de aplicación de la fuerza (en el sistema de coordenadas global) y $\mathcal{x}$ es la posición del centro de masas del objeto. Por tanto, cuanto mayor es la distancia del punto de aplicación al centro de masas, mayor será el torque y viceversa. Además, la dirección del torque es la misma que la dirección de la velocidad angular que éste provoca (Figura 9) (18).

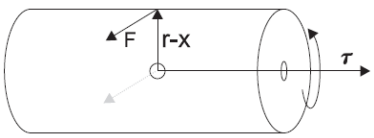


Figura 9. Torque sobre un cilindro que se le aplica una fuerza.

La tendencia de una fuerza $\mathcal{F}$ a imprimirle un movimiento de rotación a un CR alrededor de un punto, se mide mediante el concepto de momento o torque. En otras

palabras, el torque es una medida de la cantidad de rotación que una fuerza tiende a imprimirle a un CR, respecto a un punto (15).

1.5.8 Momento lineal y momento angular

El momento lineal o cantidad de movimiento de una partícula se define como el producto de la masa por su velocidad ($\boldsymbol{p_i} = \boldsymbol{m_i}\,\boldsymbol{v_i}$). En el caso de un sistema de partículas, el momento total del sistema es igual a la suma del momento de todas las partículas que lo forman (18).

$$\boldsymbol{\mathcal{P}} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{\mathcal{P}_i}$$

Ecuación 5. Momento lineal

El momento angular $\boldsymbol{\mathcal{L}_i}$ de una partícula $\boldsymbol{i}$ que describe una trayectoria curvilínea con velocidad $\boldsymbol{v_i}$, está dado por:

$$\boldsymbol{\mathcal{L}_i} = \boldsymbol{m_i}\,\boldsymbol{r_i} \times \boldsymbol{v_i}$$

Ecuación 6. Momento angular de una partícula $\boldsymbol{i}$

donde $\boldsymbol{m_i}$ es la masa de la partícula y $\boldsymbol{r_i}$ es su vector posición respecto al origen de coordenadas.

En el caso de un CR, cuando rota alrededor de un eje determinado, esta definición sigue siendo válida para cualquier partícula. Además, si los momentos angulares de todas las partículas del cuerpo se evalúan respecto al mismo punto, el momento angular total del CR está dado por:

$$\begin{aligned}\boldsymbol{\mathcal{L}_i} &= \boldsymbol{m_i}\,\boldsymbol{r_i} \times \boldsymbol{v_i} \\ &= \textstyle\sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \boldsymbol{\mathcal{L}_i}\end{aligned}$$

Ecuación 7. Momento angular total del CR

1.6 Características de los motores físicos

Un motor de simulación física simula los modelos del mundo utilizando variables como la velocidad, masa, gravedad, fuerza o rozamiento. Este tipo de simulación hace ver más real las acciones del mundo virtual, creando un mejor impacto visual.

Estos motores permiten agregar comportamiento físico a escenas tridimensionales con un costo computacional aceptable, abstrayendo al usuario de la implementación de

métodos de resolución de dinámica y detección de colisiones. Esto permite centrar el esfuerzo en otros aspectos de la aplicación y lograr un mayor realismo en las escenas a simular.

Un motor de simulación física puede abarcar una amplia gama de características y funcionalidades, aunque la mayor parte de las veces el término se refiere a un tipo concreto de simulación de la dinámica de CR. Esta dinámica se encarga de determinar el movimiento de estos CR y su interacción ante la influencia de fuerzas.

Las tres tareas principales que deben estar soportadas por un motor de simulación física son la detección de colisiones, su resolución (junto con otras restricciones de los objetos) y calcular la actualización del mundo tras esas interacciones. De forma general, las características que suelen estar presentes en motores de simulación física son (20):

- Detección de colisiones entre objetos dinámicos de la escena.
- Definición de geometría estática de la escena (formas de colisión) que formen el escenario del entorno virtual. Este tipo de geometría puede ser más compleja que la geometría de cuerpos dinámicos.
- Especificación de fuerzas (tales como viento, rozamiento, gravedad), que añadirán realismo al entorno.
- Definición de diversos tipos de articulaciones, tanto en elementos del escenario (*Ball and Socket, Hinge, Slider*) como en la descripción de las articulaciones de personajes.

A continuación se detallan algunas de las características importantes de los motores físicos.

1.7 Colisiones

En muchas aplicaciones de RV es necesario simular la interacción entre los CR. Uno de los requisitos básicos a garantizar es que no exista penetración al interactuar.

La detección de colisiones (también conocida como determinación de contacto) es un conjunto de pruebas en las cuales se determina si dos o más objetos están ocupando el mismo espacio o si están muy cercanos dado una distancia mínima.

Es frecuente establecer varios niveles de profundidad en el momento de detectar una colisión, de forma que primero se aplican las formas de colisión más generales, que requieren menos tiempo de ejecución a modo de filtro y luego se aplican las más precisas que requieren una mayor cantidad de cálculos con los objetos que no fueron descartados en niveles anteriores (22).

Para calcular las colisiones, los objetos se representan internamente por una forma geométrica sencilla (como esferas, cajas, cilindros). Esta representación interna se calcula para determinar la posición y orientación del objeto en el mundo. Estos datos, con una descripción matemática mucho más simple y eficiente, son diferentes de los que se emplean en la representación visual del objeto (que cuentan con un mayor nivel de detalle) (20).

Habitualmente se trata de simplificar al máximo la forma de colisión. Siempre será preferible emplear tipos de datos simples aunque no sean tan exactos, pero se logra una mayor eficiencia en el cálculo de las colisiones y proporciona una mayor rapidez en la simulación garantizando computacionalmente un costo bajo. La Figura 10 muestra algunos ejemplos de aproximación de formas de colisión simples para ciertos objetos.

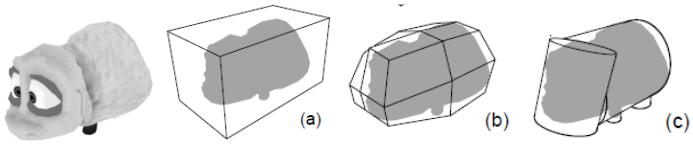


Figura 10. Formas de colisión

Las entidades de una aplicación pueden tener diferentes formas de colisión o incluso pueden compartir varias primitivas básicas (para representar por ejemplo cada parte de la articulación de un brazo robótico). Algunas de las primitivas soportadas habitualmente son:

Esferas: son las primitivas más simples y eficientes; basta con definir su centro y radio (uso de un vector de 4 elementos).

Cajas: por cuestiones de eficiencia, se suelen emplear cajas límite alineadas con los ejes del sistema de coordenadas (AABB o ***Axis Aligned Bounding Box***). Las cajas AABB se definen mediante las coordenadas de dos extremos opuestos. El principal problema de las cajas AABB es que, para resultar eficientes, requieren estar alineadas con los ejes del sistema de coordenas globales. Esto implica que si el objeto rota, como se muestra en la Figura 11, la aproximación de forma puede resultar de baja calidad.

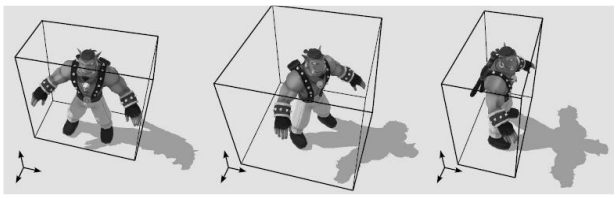


Figura 11. Forma de colisión de cajas

Por su eficiencia, este tipo de cajas suelen emplearse para realizar una primera aproximación a la intersección de objetos para, posteriormente, emplear formas más precisas en el cálculo de la colisión.

Por su parte, las cajas OBB (*Oriented Bounding Box*) definen una rotación relativa al sistema de coordenadas. Su descripción es muy simple y permiten calcular la colisión entre primitivas de una forma muy eficiente.

Cilindros: los cilindros son ampliamente utilizados. Se definen mediante dos puntos y un radio. Una extensión de esta forma básica es la cápsula, que es un cuerpo compuesto por un cilindro y dos semiesferas (ver Figura 10 (c)). Puede igualmente verse como el volumen resultante de desplazar una esfera entre dos puntos. El cálculo de la intersección con cápsulas es más eficiente que con esferas o cajas, por lo que se emplean para el modelo de formas de colisión en formas que son aproximadamente cilíndricas (como las extremidades del cuerpo humano).

Malla poligonal: en ciertas ocasiones puede ser interesante utilizar mallas arbitrarias. Este tipo de superficies pueden ser abiertas (no es necesario que definan un volumen) y se construyen como mallas de triángulos. Las mallas poligonales se suelen emplear en elementos de geometría estática, como elementos del escenario, terrenos. (ver Figura 12) Este tipo de formas de colisión son las más complejas computacionalmente, debido a que se debe probar con cada triángulo. Así, muchos juegos y aplicaciones tratan de limitar el uso de este tipo de formas de colisión para evitar que el rendimiento se desplome (20).

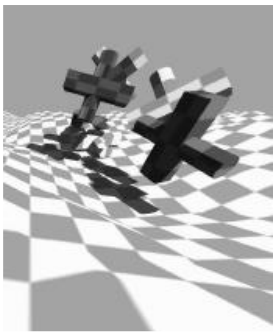


Figura 12. Mallas Poligonales

1.8 Restricciones

Las restricciones sirven para limitar el movimiento de un objeto, se usan en multitud de situaciones, como en puertas, suspensiones de vehículos, cadenas, cuerdas. A continuación se enumeran brevemente los principales tipos de restricciones soportadas por los motores de simulación física. La presencia de los siguientes elementos son denominadores comunes en todos los motores de simulación física.

1.8.1 Empalmes

Los empalmes se utilizan para conectar de alguna manera dos CR en la simulación. Estos empalmes están sujetos a restricciones, que determina la manera en que están habilitados a realizar movimientos en la simulación. Los cuerpos a los cuales conecta este empalme pueden alcanzar determinadas posiciones y orientaciones uno respecto al otro, estas posiciones las determina el tipo de empalme y las restricciones sobre el mismo.

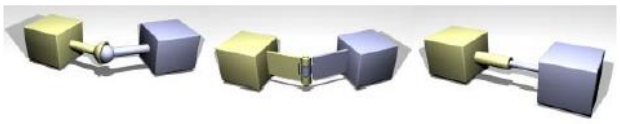


Figura 13. Tipos de empalmes.

Según el tipo de restricciones sobre el movimiento de cuerpos que se desea modelar, se debe elegir el tipo de empalme que más se adapte al mismo y ajustarlo a lo que se requiera. Algunos ejemplos de empalmes que pueden ser encontrados en motores físicos se muestra a continuación:

***Ball and Socket*:** se especifica un punto de anclaje entre ambos cuerpos y este empalme actúa de tal manera que mantiene la posición relativa de dicho anclaje con respecto a los cuerpos inalterada. Sirve por ejemplo para modelar la rotación que ocurre en el hombro de un personaje (23).

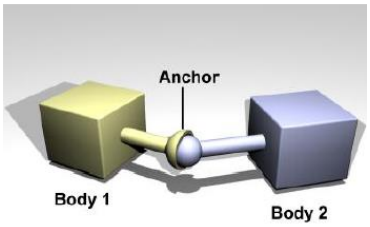


Figura 14: *Ball and Socket.*

***Hinge*:** su funcionamiento es análogo al de una bisagra. Permite a dos cuerpos rotar alrededor de un eje pero sin cambiar su distancia con respecto a dicho eje. Se utiliza para evitar, por ejemplo, que el codo de un brazo robótico adopte una posición imposible (23).

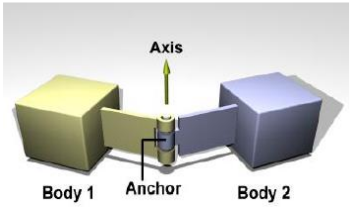


Figura 15*: Hinge.*

***Slider*:** este empalme fija ambos cuerpos entre sí y les impone la restricción de que cada uno se pueda mover en un solo eje sin afectar al otro, esto asumiendo que dicho empalme no tiene restricciones en su movimiento. En caso de que uno de los cuerpos se mueva con respecto a los otros ejes, el movimiento del otro cuerpo será el mismo con respecto a estos otros ejes (23).

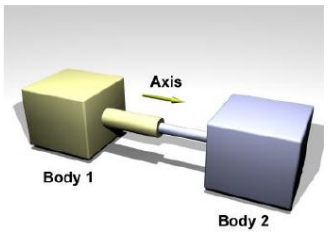


Figura 16: *Slider*.

1.9 Tipos de motores de simulación física

El desarrollo de un motor de simulación física desde cero es una tarea compleja y que requiere gran cantidad de tiempo. Actualmente, gracias al crecimiento tecnológico del hardware se han desarrollado varios motores físicos implementados en forma de bibliotecas de cálculo, tanto basados en licencias libres como comerciales. A continuación se describirán algunas de las bibliotecas más utilizadas:

Open Dynamics Engine

ODE (*Open Dynamics Engine*) es una biblioteca de software libre, de código abierto, implementada utilizando lenguaje c++. Es multiplataforma, con soporte para Linux, Windows y MacOS. Ogre cuenta con un *wrapper* para utilizar este motor de simulación física.

Características

Es buena para simular estructuras de CR articulados. Para crear estructuras de esta naturaleza se conectan CR de diferentes tamaños y formas mediante el uso de empalmes de diferentes características.

Entre sus características incluye la detección de colisiones integrada. Al ser una biblioteca de software libre, hay gran cantidad de bancos de pruebas y manuales, estando en continuo proceso de evolución y mejora.

Un punto importante de los motores físicos a tener en cuenta es su estabilidad, es decir, que lo errores de cálculo no aumenten de forma descontrolada. Según su documentación ODE tiene como característica un buen manejo de estos errores, evitando que el sistema se vuelva matemáticamente inestable sin una razón aparente, logrando esto a partir del hecho que se prioriza la estabilidad y la velocidad de la

simulación por sobre la precisión física. Las aplicaciones probadas sobre este motor incluyen la simulación de vehículos, criaturas bípedas y demás cuerpos sobre un entorno de RV.

ODE es un motor estable, pero no muy preciso, a menos que el paso de integración sea pequeño. Aunque, excepto para simulaciones de alto nivel, este motor parece físicamente perfecto (24).

Newton Game Dynamics

Esta biblioteca es de código cerrado, sin embargo es gratuita. Está implementada enteramente en lenguaje C y es multiplataforma. Tiene soporte para Linux, Windows y MacOS. Ogre cuenta con un *wrapper* para utilizar este motor de simulación física.

Características

Esta biblioteca es rápida, estable y muy sencilla de utilizar. De forma muy fácil se pueden modelar objetos complejos como muñecos con características humanas o vehículos con ruedas, manipulando todo tipo de características de los mismos.

Según su documentación, prioriza la exactitud física por sobre la velocidad en la simulación. Una característica importante es que resuelve el avance de la simulación de un modo determinista, lo que la hace una herramienta muy utilizada no solo en juegos sino también en simulaciones de situaciones reales, pues la simulación puede ser reproducible.

Una cualidad importante es el hecho de que puede manejar simulaciones en las cuales existan cuerpos que posean un muy amplio rango de masas. Es muy común en algunas bibliotecas observar que al realizar simulaciones en las cuales los cuerpos involucrados poseen masas que difieren mucho, la simulación se vuelve numéricamente inestable, observándose efectos no deseables.

Aunque no presenta gran cantidad de documentación, posee un foro muy activo en el cual los usuarios discuten dudas y problemas (25).

Physx

Esta biblioteca está programada en c++, es extensamente utilizada en el desarrollo de juegos. Es de código cerrado, con licencia comercial privativa, se utiliza de forma libre en proyectos no comerciales. Es multiplataforma, tiene soporte para Windows, Linux y Mac. Ogre cuenta con un *wrapper* para utilizar este motor de simulación física.

Características

El SDK de PhysX une simulación dinámica de CR y detección de colisiones en un paquete de fácil uso. El componente dinámico le permite simular objetos con alto grado de realismo. Hace uso de conceptos físicos como puntos de referencia, posición, velocidad, aceleración, momentos, fuerzas, movimientos rotacionales, energías, fricciones, impulsos, colisiones y uniones.

Es un potente motor de física que permite realizar en tiempo real los cálculos de física. PhysX está optimizado para acelerar por hardware el cálculo de la física a través de numerosos procesadores paralelos de alta capacidad. En la actualidad, este motor proporciona un aumento exponencial de la capacidad de procesamiento de la física.

Presenta extensa y detallada documentación, con diversos documentos y ejemplos de uso. Además posee un foro muy activo en el cual se pueden aclarar dudas y encontrar soluciones a los problemas más variados. Presenta una muy interesante combinación de velocidad, precisión, robustez y estabilidad, transformándola en uno de los motores físicos más fieles a la realidad (26).

Havok

Este es un motor físico de código cerrado con licencia comercial privativa, no es de libre uso para ningún tipo de proyecto. Es multiplataforma, tiene soporte para Windows, Linux y Mac.

Características

El motor Havok fue comprado por Intel para hacer competencia a su principal rival, Physx. Ofrece un gran rendimiento en el tiempo de detección de colisiones y simulación real de soluciones físicas. Ofrece rapidez y robustez por lo que se ha convertido en referencia dentro de la industria de los videojuegos. También ha sido elegido por los principales desarrolladores de juegos en más de 200 títulos en marcha y otros en desarrollo (27).

Bullet

Es de código abierto, multiproceso, para uso comercial de forma gratuita. Presenta un diseño modular en c++. Es multiplataforma, tiene soporte para Windows, Linux y Mac. Ogre cuenta con un *wrapper* para utilizar este motor de simulación física.

Características

El motor *Bullet* es una biblioteca profesional para la detección de colisiones y simulación de la dinámica de CR y blandos. Es muy buen motor físico, integrado en el

editor gráfico Blender. Utiliza la biblioteca ODE en algunas de sus funciones. Es sencillo de utilizar. Tiene un gran rendimiento en la detección de colisiones entre cuerpos muy complejos, posee solución de restricciones en la dinámica de CR rápida y estable, incorpora la dinámica de cuerpos deformables como ropas y volúmenes deformables, en interacción con CR (28).

PAL

La Capa de Abstracción Física (PAL) proporciona una interfaz unificada a un número de diferentes motores de simulación física. Esto permite el uso de múltiples motores dentro de una aplicación. Es compatible con una gran cantidad de motores de física.

Características

PAL tiene soporte para los siguientes motores físicos: *Bullet*, *Havok*, *JigLib*, *Newton*, ODE, *OpenTissue*, *PhysX*, *Tokamak*, *TrueAxis*. Posee en subsistema para el cálculo de las colisiones, cuenta con formas de colisión para representar a los cuerpos en el mundo físico (cajas, esferas, cápsulas, cuerpos compuestos). Tiene soporte para la mayoría de las características de los motores físicos. Está en constante actualización, tiene soporte para Linux. PAL es flexible, permite incluir física a una aplicación de una forma rápida y sencilla (29).

1.10 Arquitectura de los Laboratorios Virtuales

El término Arquitectura de Software (AS) es abordado por un gran número de autores. Cada uno aporta su punto de vista en función de su propósito, a continuación se muestran algunas definiciones que permiten una correcta interpretación del término.

Philippe Kruchten[1] refiriéndose a la AS expresa:

“La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requisitos de desempeño de un sistema, así como los requisitos no funcionales, la confiabilidad, escalabilidad, portabilidad y disponibilidad (30).”

La IEEE define: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución (31).”

La AS para los LV tiene como objetivo ser la base fundamental del desarrollo, es por ello que se puntualiza “Laboratorio Virtual” como:

[1] Director de *Rational Unified Process* (RUP)

“Espacio electrónico de trabajo concebido para la colaboración y la experimentación a distancia con objeto de investigar o realizar otras actividades creativas, elaborar y difundir resultados mediante tecnologías difundidas de información y comunicación.”

Un LV es diferente de un “laboratorio verdadero” o de un “laboratorio tradicional”. Sin embargo, no se considera que el LV vaya a suplantar a los verdaderos laboratorios o competir con ellos. En cambio, los LV constituyen una posible extensión de los verdaderos laboratorios y abren nuevas perspectivas que no se podían explorar completamente, dentro de un laboratorio verdadero, a un costo asequible (32).

Otras definiciones:

Componente: unidad de composición de aplicaciones, que posee un conjunto de interfaces especificadas contractualmente y dependencias del contexto explícitas.

Componente estático: componente que se distribuye en forma de biblioteca dinámica o estática. Es necesario hacer un enlace estático en tiempo de compilación para su uso.

Componente dinámico: componente que se distribuye en forma de biblioteca dinámica. No es necesario hacer un enlace estático para su uso (33).

1.10.1 Descripción general de la arquitectura

El propósito de la arquitectura es convertir los procesos más comunes a todos los LV en componentes. Así, es posible ensamblar un cierto número de estos y lograr la funcionalidad básica de un LV a la vez que se reducen los costos de desarrollo. Esta utiliza estilos arquitectónicos basados en capas y en componentes. La combinación de estos estilos permite un amplio nivel de reutilización y facilidad de desarrollo.

Consideraciones finales del capítulo

Se mostraron los principales factores que influyen en el desarrollo de la simulación de la dinámica de los CR brindando aspectos matemáticos y físicos necesarios que sirven para una mejor comprensión del componente a realizar.

Se investigó acerca de las características de los motores físicos implementados actualmente.

Se analizó la arquitectura que se utiliza para el desarrollo de los LV donde se pudo identificar que la base fundamental de esta es el uso de componentes para lograr las funcionalidades de los laboratorios.

Capítulo 2: Descripción de la solución técnica

Introducción

En el desarrollo de aplicaciones gráficas es de gran importancia lograr el mayor realismo con el mejor rendimiento posible. Para ello es de gran utilidad lograr la implementación y simulación de las leyes físicas en los entornos virtuales.

En el presente capítulo se plantean las soluciones técnicas para lograr satisfacer los objetivos este trabajo. Se analiza el ambiente de desarrollo a utilizar. Se realiza una comparación entre los motores de simulación física mencionados en el capítulo anterior y se selecciona el que mejor se adapte a las necesidades del proyecto.

Se analizan las características del sistema a realizar, 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 componente dinámico a realizar. Se enumeran y detallan los requisitos funcionales y no funcionales, que permiten tener una concepción general del componente, 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 Selección del ambiente de desarrollo para el componente

Luego de la fundamentación teórica realizada se procede a la descripción de las herramientas que serán empleadas para el desarrollo del componente dinámico:

- La metodología de desarrollo a emplear será RUP, también conocido como Proceso Unificado, no es una metodología con pasos permanentes y rígidos, es adaptable al contexto y a las necesidades de cada proyecto. RUP trabaja conjuntamente con UML y es la metodología estándar y más utilizada para análisis, diseño, implementación y documentación de desarrolladores orientados a objetos.
- Como herramienta *CASE* se selecciona *Visual Paradigm for UML* Versión 8.0 pues soporta el ciclo completo de desarrollo de software y ofrece un conjunto de herramientas necesarias para la realización de las diferentes tareas del proceso.
- Para la implementación del componente se utiliza el lenguaje c++, el cual es uno de los más empleados en la programación de gráficos por computadora. Entre las características más importantes se destacan: robusto, orientado a objetos, permite la reutilización de código mediante la herencia, la sobrecarga de operadores,

funciones, es libre y además es el que se utiliza en el proyecto para el desarrollo de los LV.

- Como entorno de desarrollo integrado se emplea Microsoft Visual Studio herramienta poderosa, fuerte y voluminosa. Tiene varios lenguajes entre ellos se encuentra c++.
- Para el renderizado de las escenas de los LV se emplea el motor gráfico Ogre3D, utiliza c++, es multiplataforma, soporta las API gráficas OpenGL y DirectX y es de código abierto. Es el motor gráfico utilizado en el proyecto.

2.2 Comparación de los motores de simulación física

Existen características importantes de los motores físicos que es necesario analizar para tomar correctas decisiones en la elección de alguno de ellos en el marco de este proyecto.

El propósito de este estudio es evaluar y comparar los motores físicos descritos en el capítulo 1. Es importante considerar que estas evaluaciones están basadas en su mayoría en los CR siendo este el objeto de estudio de la investigación. Los motores físicos modernos incorporan ciertos elementos complejos como simulación de telas, cuerdas, pelo o fluidos. Estos elementos no serán evaluados pero si se tendrán en cuenta durante la comparación teórica. Solo se tendrán en cuenta los motores físicos que posean licencias gratuitas para uso comercial y sean multiplataforma.

Se establecieron dos parámetros como base en la evaluación.

Las características: se necesita conocer cuáles son las principales características de los motores físicos: ¿Cuáles son los tipos de empalme que soporta? ¿Cuáles son las formas de colisión que usan? ¿Soporta objetos deformables? ¿Soporta "*Vehículo*" o cualquier otra funcionalidad especial? Así como otras funcionalidades que se pueda aportar.

La documentación: la documentación es muy importante para poder entender como funciona el motor físico, así como para saber como utilizar sus funcionalidades para poder lograr los objetivos que se tracen. Es importante saber de qué manera brinda la documentación, si posee ejemplos implementados para su mejor comprensión, cuán grande es su comunidad de desarrollo entre otros aspectos.

A continuación se muestran comparaciones entre cada motor en cuanto a sus características:

Empalmes /Motor	*Bullet*	*ODE*	*Tokamak*	*Newton*	*Jiglib*	*Open Tissue*
Ball and Socket	*SI*	*SI*	*SI*	*SI*	*SI*	*SI*
Hinge	*SI*	*SI*	*SI*	*SI*	*SI*	*SI*
Hinge-2	*NO*	*SI*	*NO*	*NO*	*SI*	*NO*
Slider	*SI*	*SI*	*NO*	*SI*	*NO*	*SI*
Universal	*NO*	*SI*	*NO*	*SI*	*NO*	*SI*
Corkscrew	*NO*	*NO*	*NO*	*SI*	*SI*	*NO*
Fixed	*NO*	*SI*	*NO*	*NO*	*SI*	*NO*
Vehículo	*SI*	*NO*	*NO*	*SI*	*SI*	*NO*
Evaluación de Soporte	*Regular*	*Bueno*	*Malo*	*Bueno*	*Bueno*	*Regular*

Tabla 1. Tipos de empalmes que admite cada motor. La fila de vehículos en la tabla indica si los vehículos son apoyados de forma nativa por el motor.

Formas de colisión / Motor	*Bullet*	*ODE*	*Tokamak*	*Newton*	*Jiglib*	*Open Tissue*
Caja	*SI*	*SI*	*SI*	*SI*	*SI*	*SI*
Esfera	*SI*	*SI*	*SI*	*SI*	*SI*	*SI*
Cápsula	*SI*	*SI*	*SI*	*SI*	*SI*	*NO*
Cilindro	*NO*	*SI*	*NO*	*SI*	*NO*	*SI*
Volumen Convexo	*SI*	*SI*	*SI*	*SI*	*NO*	*NO*
Malla Poligonal	*SI*	*SI*	*SI*	*SI*	*SI*	*SI*
Objetos Compuestos	*SI*	*SI*	*SI*	*SI*	*SI*	*SI*
Soporte para Cuerpos Blandos	*SI*	*NO*	*NO*	*NO*	*NO*	*NO*
Cuerpos Deformables	*SI*	*NO*	*SI*	*NO*	*NO*	*SI*
Evaluación de Soporte	*Bueno*	*Bueno*	*Bueno*	*Bueno*	*Regular*	*Regular*

Tabla 2. Formas de colisión que admite cada motor y funcionalidades adicionales que soporta.

Coeficiente de fricción /Motor	*Bullet*	*ODE*	*Tokamak*	*Newton*	*Jiglib*
Estático	*SI*	*SI*	*SI*	*SI*	*SI*
Dinámico	*NO*	*SI*	*NO*	*SI*	*SI*
Resistencia	*SI*	*SI*	*SI*	*SI*	*NO*
Evaluación de Soporte	*Regular*	*Bueno*	*Regular*	*Bueno*	*Regular*

Tabla 3. Tipos de materiales que admite cada motor**.**

2.2.1 Resultados de la comparación realizada

En general todos los motores físicos poseen las formas de colisión básicas (caja, esfera, cápsula o cilindro), en cuanto a los empalmes *Tokamak* no posee buen soporte pues solo incluye *Ball and Sokect* y *Hinge,* pero en general los otros motores poseen una buena variedad de empalmes y al menos cada motor posee dos de las propiedades de los coeficientes de fricción.

Entre los más completos se encuentra ODE y Newton pues tienen la mejor evaluación en las tres tablas comparativas es decir tienen soporte para la mayoría de las funcionalidades.

Cuando se analiza la documentación, *Open Tissue* tiene muy poca información disponible en su sitio oficial. ODE realmente tiene buena documentación con muchos ejemplos implementados donde los usuarios pueden encontrar las funcionalidades en uso. *Newton, Tokamak, Bullet* y *Jiglib* también tiene buena documentación pero no tan extensiva como ODE.

Otro aspecto importante es que ODE, *Bullet, Newton* y *Open Tissune* son motores de código abierto.

2.3 Elección del motor de simulación física

Entre los diversos motores físicos que podrían llegar a cumplir los requisitos de este proyecto, en primer lugar se analizó la biblioteca *Newton Game Dynamics* por las características que posee y su variedad en las funcionalidades. Fundamentalmente la flexibilidad en el manejo del paso del tiempo y la facilidad de uso la transformaba en una interesante alternativa. Esta biblioteca tiene como desventaja que prioriza la exactitud física sobre la velocidad en la simulación.

Las siguientes alternativas que se evaluaron fueron la utilización de las bibliotecas ODE y *Bullet*.

ODE es un motor físico que no presenta tanta flexibilidad en el manejo de errores del paso del tiempo como la anterior alternativa, pero presenta diversas cualidades que implican que esta sea una alternativa a tener en cuenta. Posee una buena documentación, incluye la detección de colisiones integrada, tiene buen manejo de errores de cálculo, es buena para simular estructuras de CR articulados.

Bullet es una biblioteca sencilla de utilizar, permite la simulación de la dinámica de CR y blandos. Tiene un gran rendimiento en la detección de colisiones entre cuerpos muy complejos.

La elección de un motor de simulación física en específico se convierte en una tarea muy compleja debido a la diversidad y comportamiento similar de cada motor. Cada uno posee un conjunto de características robustas descritas anteriormente. Ocurre que los LV son de diferentes materias lo que provoca variedad; por ejemplo, en unos se necesitará conseguir mayor velocidad de cálculo, mientras que en otros de carácter científico primará la precisión.

En la investigación se encontró una solución que plantea la implementación de una capa de abstracción que permite comunicar un motor gráfico con distintos motores físicos llamada PAL. Esta capa de abstracción facilita el intercambio de motores físicos de una manera transparente y cómoda para el usuario proponiendo una interfaz unificada para el uso de estos motores.

Después de haber analizado los principales motores físicos existentes se decidió utilizar PAL. Esta capa de abstracción ofrece una serie de beneficios a los desarrolladores. Se puede integrar fácilmente, no restringe el uso de un motor de física en particular, esto le da más flexibilidad lo que permite actualizar fácilmente el sistema de física. Se puede seleccionar diferentes motores para plataformas alternativas o el intercambio a otro motor si los desarrolladores dejan de darle apoyo a su motor. Esta flexibilidad permite elegir el motor que mejor rendimiento tenga y el que mejor se adapte a las necesidades de la aplicación.

2.4 Capa de abstracción física

Hoy en día un motor de física se utiliza ampliamente en el área de simulación en 3D y existen muchas opciones de este tipo de producto. Cada uno de ellos tiene un carácter común en sus funcionalidades y tienen tanto ventajas como desventajas que representan su singularidad. Básicamente, si se quisiera utilizar un motor de física, se

necesita estudiar su propia API[2]. Por ejemplo, si se necesita una caja en el mundo de física, el siguiente ejemplo muestra como funciona y se define en los siguientes motores físicos:

***Novodex*:**

```
m_pBoxShape = new NxBoxShapeDesc();

m_pBoxShape->dimensions = NxVec3 (dim.x*0.5f, dim.y*0.5f, dim.z*0.5f);
```

***Bullet*:**

```
pbtBoxShape = new btBoxShape (btVector3 (dim.x, dim.y, dim.z));
```

***Havok*:**

```
hkVector4 boxSize(dim.x *0.5, dim.y *0.5, dim.z *0.5);

pShape = new hkpBoxShape(boxSize);

BuildBody(dim.x, dim.y, dim.z, mass, true);
```

Los parámetros que se necesitan para crear una caja en cada motor son las tres dimensiones de la caja (la posición de la caja no se tuvo en cuenta para este ejemplo). Se necesitó escribir tres líneas de código diferente para implementar esa misma función.

Cuando se implementan aplicaciones que utilizan motores físicos, los desarrolladores se encuentran con el problema que cada motor presenta una interfaz distinta en sus métodos y en la mayoría de los casos se debe reescribir el código que se ha programado si se desea intercambiar un motor por otro.

Por esto es necesario proveer una arquitectura de clases que brinde una interfaz uniforme de comunicación con el conjunto de motores físicos a utilizar, de manera que no sólo se permite reutilizar código escrito previamente (independiente al motor físico que se utilice) sino que se gana extensibilidad y adaptabilidad en el sistema. La capa de abstracción física se implementó utilizando un conjunto de patrones de diseño, principalmente el patrón fábrica abstracta.

Este patrón permite la creación de un objeto sin especificar la clase exacta del objeto que se va a crear. De esta manera se puede pedir a la fábrica de PAL crear un objeto "Caja" y esta se encarga de verificar a qué tipo de clase se hace referencia, es decir

[2] **Interfaz de programación de aplicaciones** (**IPA**) o API (del inglés *Application Programming Interface*) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

crear una caja de ODE o una caja de *Bullet* en dependencia del motor que se esté utilizando.

Para seleccionar el motor de física que se desea utilizar en PAL en primer lugar se debe llamar mediante un objeto de la fábrica abstracta el método correspondiente a esta acción en este caso sería **`PF->SelectEngine()`** y se le pasa el nombre del motor de física que desea utilizar por parámetro.

Después de esto se puede crear cualquier objeto mediante una llamada al método apropiado. En este caso: **`PF->createBox()`**. Como alternativa, puede crear directamente a través del nombre del objeto proporcionado a la fábrica: **`PF->CreateObject("Palbox").`**

El patrón de diseño fábrica abstracta intenta solucionar los problemas que se presentan al crear diferentes familias de objetos (en este caso, una familia de objetos por cada motor físico que se desea utilizar). Cada motor físico tiene sus parámetros y características, con lo cual el código de ejecución escrito para utilizar un motor en particular (por ejemplo *Newton Game Dynamics*), no servirá para utilizar otro como *Open Dynamics Engine*, con lo cual es necesario contar con un mecanismo que permita reutilizar el código escrito. Dicho mecanismo es provisto por el patrón de diseño fábrica abstracta, mediante la definición de interfaces genéricas para la creación y manipulación de los distintos objetos de los motores físicos.

Es prioritario que sólo exista una única fábrica en ejecución, de modo que no existan usos inconsistentes de los objetos de la simulación. Para esto se añade a la solución propuesta por el patrón fábrica abstracta el uso del patrón de diseño *Singleton* el cual está diseñado para restringir la creación de objetos pertenecientes a una clase a un único objeto, garantizando que una clase sólo tenga una instancia y proporcione un punto de acceso global a ella.

La figura 17 muestra una representación de las clases principales de PAL:

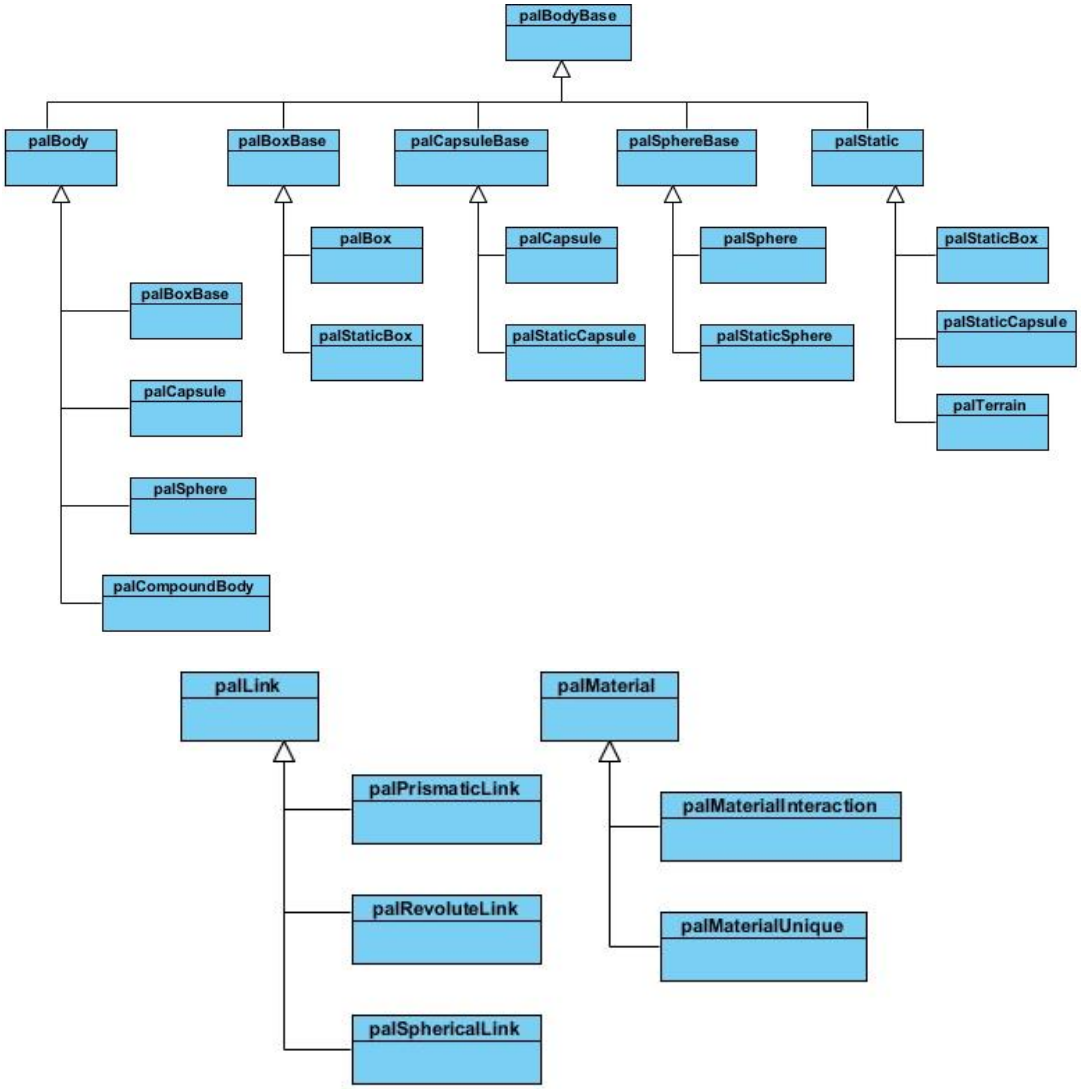


Figura 17. Principales clases de PAL

Se tiene una clase ***palBodyBase*** que representa un objeto en el motor físico. Este objeto puede estar formado por varias formas que pueden ser simples como cajas, esferas, cápsulas, cilindros o más complejas como mallas de triángulos, puede ser también una forma compuesta por varias simples. El objeto tiene localización y masa. Cada forma puede tener asociado un material. También se puede tener empalmes de diferentes tipos entre dos objetos (por ejemplo la articulación de un brazo de robot).

2.5 Utilización de PAL

2.5.1 Inicialización y configuración de PAL

Lo primero que se debe hacer para empezar a utilizar las funcionalidades de PAL es incluir la biblioteca física a través del archivo "palFactory.h". Luego se carga el motor físico apropiado y se selecciona.

`PF->LoadPALfromDLL("D:…/pal/bin");` carga las bibliotecas del motor físico que se utiliza, que deben encontrarse en la dirección que se pasa por parámetro.

`PF->SelectEngine("Nombre del Motor Físico, Ej: ODE")` selecciona el motor de física para ejecutar la simulación.

En este punto, se ha completado la inicialización de PAL.

Inicialización del motor físico:

El siguiente paso es inicializar el motor físico. Primero se crea una instancia de la clase ***palPhysics*** y se llama al método que inicializa los valores de la gravedad:

```
palPhysics *pp = PF->CreatePhysics();

pp->Init(0,-9.8f,0);.
```

2.5.2 Creación de objetos

Hay varios tipos de objetos predefinidos por PAL y cada uno tiene propiedades diferentes. Cada objeto interactúa con el entorno de forma distinta, dependiendo de las fuerzas que se le aplican, de las colisiones a las que se ve sometido, de las restricciones que posee, del tipo de material que tiene. En cada ejecución de cálculo de físicas, el motor deberá calcular la nueva posición y orientación del objeto teniendo en cuenta todos estos factores para la siguiente actualización.

La forma de un objeto puede contener varias partes (pues hay objetos formados por varias piezas de formas simples o complejas unidas) por lo que se debe definir cada una de estas partes con otro objeto. Algunos de los tipos de objetos que posee PAL son los siguientes (ver Figura 18):

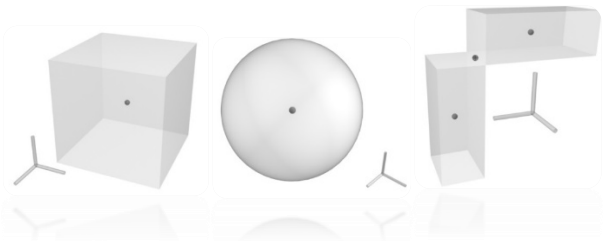


Figura 18. Tipos de Objetos Soportados por PAL.

Cajas (palBox): esta clase representa una caja que tiene una longitud (ancho, largo, alto) y una posición (x, y, z) que corresponde a la del centro de la caja.

Esferas (palSphere): esta clase representa una simple esfera en una posición(x, y, z) dada, con un radio (r) y masa (m) dada.

Cápsulas (palCapsule): esta clase representa una cápsula en una posición (x, y, z) dada, con un radio (r) dado, la longitud (l) y la masa (m).

Estos descriptores contienen funciones específicas para cada forma, de manera que el cálculo de posición y colisiones es mucho más eficiente.

Cuerpos Compuestos (palCompoundBody): esta clase representa un cuerpo compuesto que lo constituye un cuerpo con múltiples formas, esto representa un número determinado de formas básicas que se combinan para crear una forma más compleja.

2.5.3 Funciones asociada a un objeto

Algunas funciones interesantes que ofrece la clase palBody para modificar varios parámetros del objeto cuando el motor no está realizando ningún cálculo de físicas, son las siguientes:

Asignarle un grupo (**`setGroup`**, **`getGroup`**): se pueden crear grupos de formas y por ejemplo, hacer que solo colisionen entre ellos las formas de un mismo grupo o ejecutar una función cuando un grupo atraviesa un plano.

Obtener la posición y orientación del cuerpo (**`GetLocationMatrix`**): recupera la posición y orientación del cuerpo como una matriz de transformación 4x4.

Modificar la velocidad (**`setLinearVelocity`**, **`setAngularVelocity`**): modifica la velocidad lineal o angular para el siguiente cálculo de física.

Modificar material (**`SetMaterial`**): modifica el tipo de material de un objeto.

Aplicar una fuerza (**`ApplyForce`**, **`ApplyForceAtPosition`**, **`ApplyTorque`**): hay dos tipos de fuerzas dependiendo si se quiere aplicar una fuerza en un instante de tiempo (algo parecido a dar un golpe al objeto) o que el objeto se vea sometido a una fuerza constante (como ir empujando el objeto). La función **`ApplyForce`** aplica una fuerza instantánea a calcular en el siguiente proceso de física mientras que **`ApplyTorque`** aplica una fuerza continua que deberá ser aplicada antes de cada cálculo de física. Si por error se usa **`ApplyForce`** en cada ejecución del bucle del programa se apreciaría que el objeto va ganando velocidad de forma cuadrática. Estas funciones aplican la fuerza sobre el centro de masa, pero también es posible hacerlo sobre un punto en concreto del objeto con funciones como **`ApplyForceAtPosition`**.

Operaciones para desactivar un objeto (**`SetActive`**): es necesario que los objetos puedan “dormirse” (dejar de moverse y de colisionar entre ellos) si no hay fricción en la

escena o hay muchos cuerpos, pues sino el objeto estaría moviéndose infinitamente aunque fuera cada vez a menor velocidad.

2.5.4 Pintar un objeto en la escena

Normalmente se tiene una representación del objeto en el motor gráfico que corresponderá al que se debe representar en la escena y otra más simplificada para el cálculo de físicas que representara el objeto en el mundo físico.

Antes de dibujar el objeto se debe posicionar en la escena. Cuando se crea el objeto se pasa por parámetro la posición inicial **`(x, y, z)`**, también se puede modificar su posición y orientación con el método **`SetPosition(x, y, z, roll, pitch, yaw)`**

Luego se añade un terreno, vamos a utilizar un plano simple, que hace referencia a la clase **`palTerrainPlane`** de PAL.

Para construir un plano se le pasa la posición donde está ubicado y el tamaño que tendrá.

```
palTerrainPlane *pt= PF->CreateTerrainPlane();

pt->Init(0,0,0,50.0f);
```

El plano es inicializado en el origen de coordenadas (0, 0, 0) y se le pasa un tamaño de 50x50 unidades.

A continuación se adiciona el objeto en la escena:

```
palBox *pb = PF->CreateBox();

pb->Init(0,5,0, 1,1,1, 1);
```

Esto crea una caja en la posición **`(0, 5, 0)`**, cinco unidades arriba de la posición del terreno y las dimensiones serán 1x1x1 unidades, con una masa de 1 kilogramo.

Ya se tiene una escena física completa donde hay un cubo que caerá en un plano. Solo falta llamar a la función que actualiza la posición de los objetos en la escena (**`Update(0.02f)`**). En la Figura 19 se muestra un ejemplo de cubos cayendo por gravedad y colisionando entre ellos.

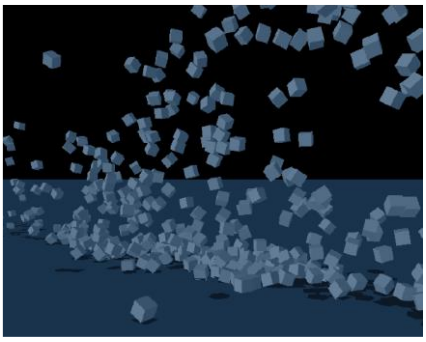


Figura 19. Cubos cayendo en una escena tridimensional.

2.5.5 Empalmes

Para crear un brazo robótico, un personaje o cualquier otro mecanismo articulado es necesario definir empalmes entre los diferentes componentes. El proceso para crear una unión es el siguiente: crear los dos objetos sometidos a la unión y declararlo en la escena.

Los empalmes que proporciona PAL son los siguientes:

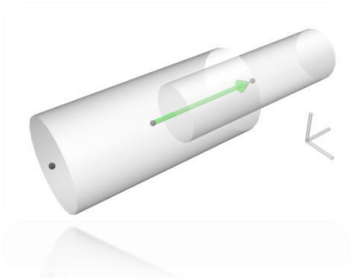


Figura 20. *PrismaticLink*.

palPrismaticLink: es una unión traslacional de un cuerpo que puede moverse a través de otro por una recta definida por un punto y un vector.

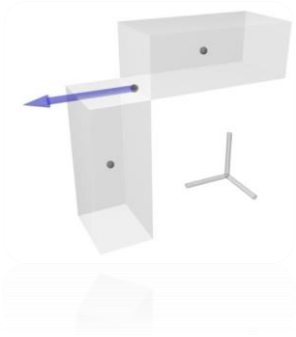


Figura 21. *RevoluteLink***.**

palRevoluteLink : define una unión de dos cuerpos por un punto y con un grado de libertad. Se debe definir el eje de rotación (vector perpendicular al plano en que

rotarán los objetos) y el punto de unión. Es útil para crear articulaciones, brazos de robot y bisagras.

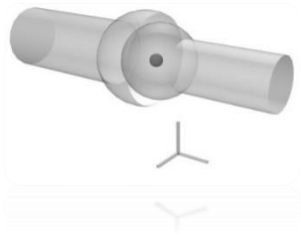


Figura 22. *SphericalLink*

palSphericalLink: unión esférica entre dos formas por un punto. Tiene tres grados de libertad: puede rotar sobre los tres ejes, por lo que también puede girar sobre sí mismo. Es necesario definir la posición y un eje global. Se pueden crear articulaciones como la del brazo con el hombro.

2.6 Solución a implementar

Debido a que la solución a implementar va a formar parte de los LV y estos poseen una arquitectura basada en capas y en componentes, se implementa un componente dinámico que utiliza PAL para incorporar física a los entornos virtuales y se inserta como una funcionalidad básica más en los LV (ver Figura 26). Entre las características fundamentales se pueden resaltar las siguientes:

- Permite incluir la simulación de las leyes físicas fundamentales en los entornos virtuales logrando una mayor inmersión y credibilidad en las escenas simuladas
- Brinda una interfaz de acceso común a los diferentes motores físicos y sus funcionalidades.
- Permite cambiar de usar un motor físico a otro e incluir física a su aplicación reduciendo los costos de desarrollo.
- Es capaz de añadir diferentes objetos de diferentes formas y estos puedan interactuar entre sí.
- Es capaz de modificar o configurar parámetros en el motor de física tales como la gravedad, coeficientes de rozamiento asociados a un material.
- Permite la comunicación fácilmente de un motor gráfico con diversos motores de simulación física, minimizando el esfuerzo de cambio y fomentando la reutilización de código.
- Brinda una interfaz que permite al desarrollador utilizar todas las funcionalidades que desee sin necesidad de tener conocimiento de la implementación.

El componente provee una interfaz independiente de la capa de abstracción física, sin agregar sobrecarga de tiempo, reduciendo significativamente el esfuerzo del programador para portar sus aplicaciones a otros motores y agregarle comportamiento físico al entorno virtual. Esto es importante, dado que cada motor físico brinda a las simulaciones velocidad de cálculo, estabilidad o precisión. Un motor físico es una solución aproximada para representar una escena del mundo real y ninguno se adapta a todas las necesidades de los usuarios.

En la Figura 23 se muestran las diferentes capas de la arquitectura de los LV. La capa de soporte contiene los componentes dinámicos que sirven de apoyo para realizar un ejercicio de un LV determinado. El control de estos componentes radica en la capa lógica, por lo que el componente dinámico a implementar debe incluirse en la capa de soporte.

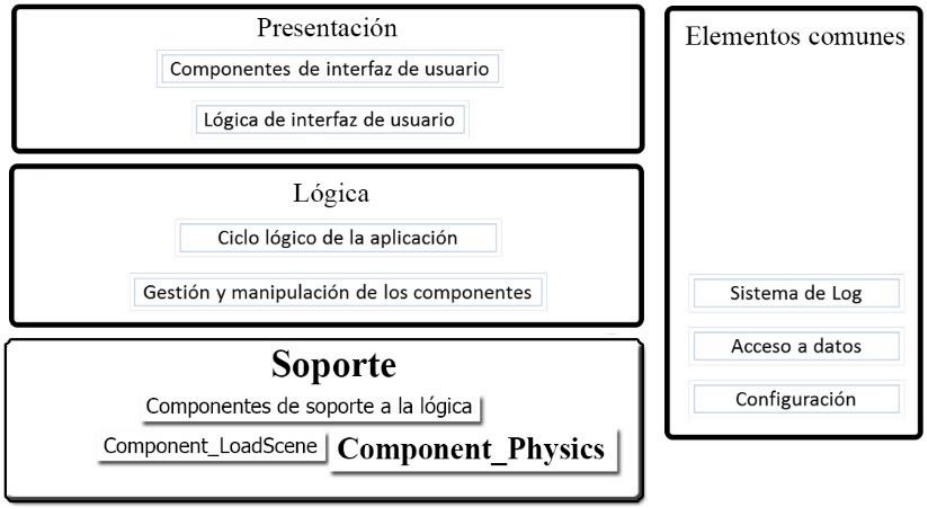


Figura 23. Capas de la Arquitectura de los LV.

2.7 Modelo de Dominio

Como parte del proceso de conceptualización del problema se realiza un modelo de domino que contribuye a comprender mejor la estructura y los conceptos asociados al funcionamiento del sistema.

El modelo de domino es un diagrama de UML, que permite mostrar los principales conceptos y relaciones que intervienen en el dominio del problema. Captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde se encuentra el sistema. Se muestra en la Figura 24 el diagrama de clases del Modelo de dominio.

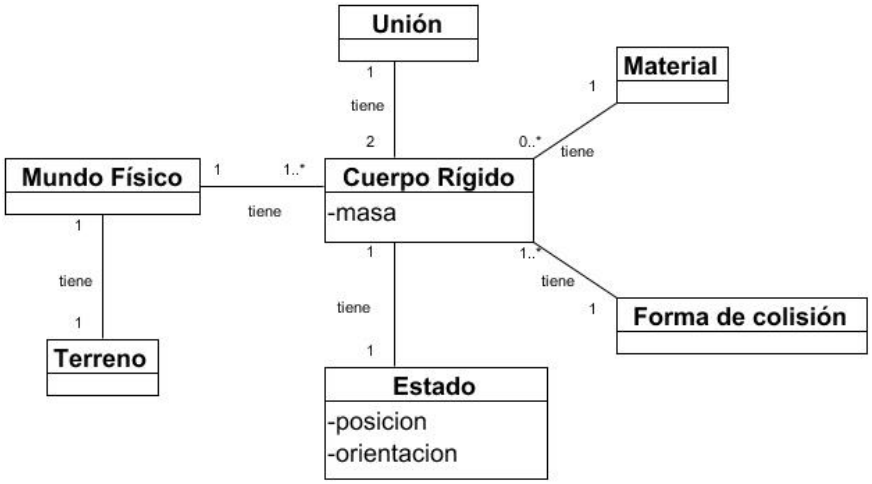


Figura 24. Modelo de Dominio**.**

Glosario de Términos

Es muy importante conocer el significado de los conceptos enunciados en el modelo anterior, a continuación se especifican los mismos.

Cuerpo Rígido: objeto de la escena que no admite deformación en su estructura y que tiene propiedades físicas por la que se regirá su movimiento como pueden ser masa y posición. Estructura que contiene y describe las propiedades físicas de un objeto en el "mundo real".

Mundo Físico: es un contenedor para todos aquellos elementos físicos que se deseen simular. Todo elemento que se simule tiene que ser añadido al Mundo y este a su vez tiene control sobre ellos. Todos los elementos en un Mundo Físico existen el mismo instante de tiempo.

Unión: es la relación existente entre dos cuerpos tales que cada uno de ellos pueden tener solo ciertas posiciones y orientaciones relativas al otro.

Forma de Colisión: son los elementos fundamentales en el sistema de colisión. Puede representar a un cuerpo físico (caja, esfera, cápsula) o puede representar un grupo de otras formas de colisión. Es el elemento físico que contiene los cuerpos en el Mundo Físico y permite las colisiones entre ellos.

Material: Superficie de un material que simula los materiales existentes en la realidad (Madera, Hielo, Hierro).

Estado: representación de atributos que definen la información espacial del cuerpo.

2.8 Reglas del negocio

- Los cuerpos que se tendrán en cuenta son los llamados CR.

- PAL soporta un gran número de motores físicos y posee una interfaz única para su comunicación, el componente a implementar brinda características que no necesariamente están soportadas por todos los motores físicos, por lo que se puede encontrar motores que soporten determinadas características y otros no.
- Los volúmenes de colisión que se tendrán en cuenta para los cuerpos son:
  - Caja
  - Cápsula
  - Esfera
  - Cuerpos compuestos
- Sólo se tendrán en cuenta como fuerzas incidentes en el cuerpo:
  - Fuerza de gravedad
  - Fuerza de rozamiento
  - Fuerza normal
  - Fuerza aplicada por el programador

  Cualquier otra fuerza que pueda incidir sobre el cuerpo en la realidad aquí no será simulada.
- Para el cálculo de la fuerza de rozamiento actuando sobre un cuerpo se tendrá en cuenta el coeficiente de fricción del cuerpo, permitiendo introducir como propiedades del material de un cuerpo:
  - Coeficiente de fricción estático
  - Coeficiente de fricción dinámico
  - Coeficiente de resistencia
- Se considera 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 considera como una masa puntual, el punto de referencia del cuerpo concuerda en todos los casos con el centro de masa y el centro de gravedad del mismo.
- Las restricciones entre dos cuerpos a tener en cuenta son:
  - *Ball and Socket*
  - *Hinge*
  - *Slider*

2.9 Especificación de los requisitos de software

2.9.1 Requisitos Funcionales

Una vez conocidos los conceptos asociados al objeto de estudio, se puede empezar a analizar ¿Qué debe hacer el sistema para que se cumpla el objetivo planteado al inicio de este trabajo?, para ello se enumera a través de los requisitos funcionales las funciones que el sistema deberá ser capaz de realizar. Dentro de ellos se incluyen las acciones que podrán ser ejecutadas por el programador y las acciones ocultas que debe realizar el sistema. De acuerdo con los objetivos planteados el sistema debe ser capaz de:

1. Configurar el mundo físico.
   1.1. Seleccionar el motor de simulación física a utilizar.
   1.2. Asignar gravedad al mundo creado.
   1.3. Asignar el “paso” de la simulación manualmente (valor fijo).
   1.4. Limpiar la física del mundo creado: Dejar de utilizar el motor físico.
2. Asignar a cada objeto de la escena su representación en el mundo físico.
3. Determinar el tipo y las constantes físicas del cuerpo.
   3.1. Especificar la forma de colisión.
   3.2. Asignar valor de la masa del cuerpo.
   3.3. Asignar dimensiones del volumen de colisión.
4. Determinar el estado inicial del cuerpo.
   4.1. Asignar posición inicial.
   4.2. Asignar orientación inicial.
5. Actualizar el estado del cuerpo.
6. Adicionar material a un cuerpo
   6.1. Crear un tipo material introduciendo el coeficiente de rozamiento estático, dinámico y de resistencia.
   6.2. Crear tipos de interacción entre dos cuerpos con materiales distintos
7. Adicionar Fuerzas a un cuerpo.
   7.1. Adicionar una fuerza al cuerpo que actúe de forma permanente sobre su centro de masa.
   7.2. Adicionar una fuerza al cuerpo que actúe sobre un punto distinto de su centro de masa.
   7.3. Adicionar una fuerza al cuerpo que actúe en un instante de tiempo sobre su centro de masa.
   7.4. Adicionar una fuerza al cuerpo que actúe en un instante de tiempo sobre un punto.

7.5. Adicionar un torque al cuerpo que actúe en un instante de tiempo.

8. Crear superficie física.
   8.1. Asignar material a la superficie física.
9. Crear unión entre cuerpos
   9.1. Establecer punto de unión entre los cuerpos.
   9.2. Especificar ángulo de movimiento de la unión.
   9.3. Definir el eje de rotación de la unión.

2.9.2 Requisitos no funcionales

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

Apariencia o interfaz externa.

- **Usabilidad:** los usuarios que trabajen con el componente deben tener conocimientos básicos de física y de toda la terminología a fin. El sistema debe estar concebido para mostrarle al programador todas las funcionalidades que brinda el sistema desde una interfaz. 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 con bajo costo de desarrollo.
  - El producto debe estar concebido para ser acoplado a la arquitectura de los LV, de tal manera que se use en caso de que el desarrollador lo necesite y no como parte indispensable de la arquitectura.
- **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 del componente dinámico se realiza en c++.
- **Legales:** se regirá por las normas ISO 690.
- **Software:** se utiliza el IDE Microsoft Visual Studio Profesional 2008 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.

Modelo de casos de uso del sistema

Utilizando las facilidades que brinda el UML, se pueden capturar los requisitos funcionales del sistema y representarlos mediante un diagrama de casos de uso. Para

ello se define cuáles 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 componente interactúa un sólo actor que se define a continuación:

Actores	Justificación
Programador	Este componente físico es un proceso común de todos los LV y está destinado a ser utilizado por programadores para incorporarle física a los LV, por tanto el programador es el que se beneficia con las funcionalidades que brinda el componente, 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 4. Actores del Sistema.

Diagrama de casos de uso

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

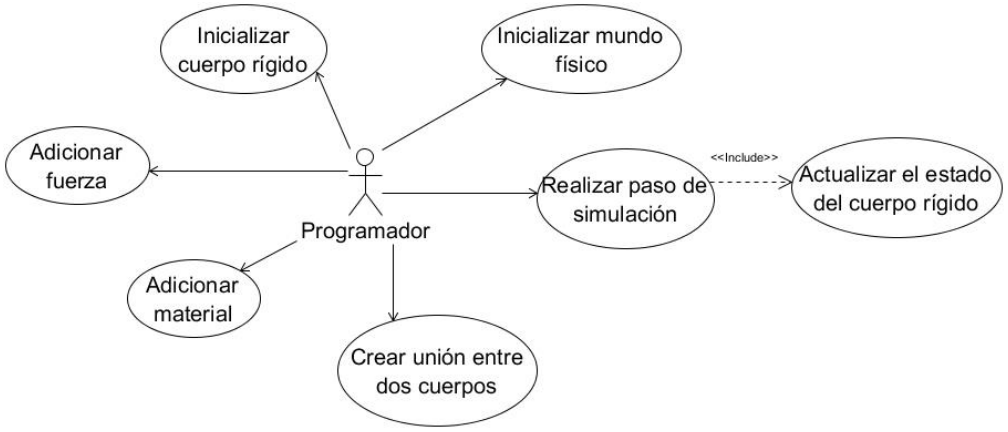


Figura 25. Diagrama de Casos de Uso del Sistema.

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 5. 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
Resumen: El caso de uso se inicia cuando el programador solicita crear el mundo físico. Se crea el mundo físico y se introduce el nombre y la dirección donde se encuentra el motor físico a utilizar, se especifica el valor de la gravedad. A continuación se especifica el tamaño del paso del tiempo de la simulación. El caso de uso finaliza cuando estas propiedades son inicializadas y se eliminan todas las configuraciones y objetos creados en el mundo físico es decir se libera toda la memoria utilizada por el motor físico.	
Referencias	RF-1.1 al 1.5
Acción del actor	Respuesta del sistema
1- Solicita insertar la dirección donde se encuentra el motor físico a utilizar.	2- Se verifica que en la dirección introducida se encuentran las bibliotecas correspondientes a un motor físico.
3- Solicita introducir el nombre del motor físico.	4- Se carga el motor físico y se crea la física del mundo.
5- Solicita la inserción del valor de la gravedad al mundo.	6- Se le asigna el valor de la gravedad al mundo. 6.1- Si no se introduce ningún valor se asumen los valores estándares de la gravedad(9.81 m/s^2)
7- Se solicita especificar el tamaño del paso de la simulación.	8- Se asigna el valor.
9- Solicita liberar toda la	10- Se eliminan todas las configuraciones y objetos

memoria utilizada por el motor físico.	creados en el mundo físico.
Precondiciones	-
Poscondiciones	Mundo físico creado con todas sus propiedades.

Tabla 6. Caso de Uso expandido: Inicializar CR.

Caso de uso	
CU-2	Inicializar CR.
Propósito	Especificar la forma de colisión se desea crear. Inicializar la posición y orientación del cuerpo creado.
Actores	Programador
Resumen: El caso de uso se inicia cuando el programador solicita crear una nueva forma de colisión. Inicializa la posición, orientación, dimensiones de la forma de colisión del cuerpo y su masa. El caso de uso finaliza cuando se le asignan los valores iniciales a al CR.	
Referencias	RF-3.1, 3.2, 3.3, 4.1, 4.2
Acción del actor	Respuesta del sistema
1- Solicita crear una forma de colisión e inicializar todas las propiedades físicas del CR, se introducen los siguientes datos: • Posición • Dimensiones • Orientación • Masa	2- Se crea la forma de colisión y se le asigna el valor inicial de la posición, dimensión, orientación y la masa del CR.
	3- Se calcula la matriz de orientación, en ella se encuentra la posición y rotación del cuerpo.
Precondiciones	Tiene que existir un objeto con estado físico asociado

	para asignarle un volumen de colisión y determinar su matriz de orientación.
Poscondiciones	Queda calculada la matriz de orientación y es asociada a un cuerpo.

Tabla 7. Caso de Uso expandido: Adicionar fuerza.

Caso de uso	
CU-3	Adicionar fuerza.
Propósito	Adicionar una fuerza al cuerpo
Actores	Programador
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	RF-7.1 al 7.5
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.
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,	2- Se adiciona la fuerza al cuerpo.

introduciendo: • 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.
Sección5: Adicionar torque al cuerpo.	
1- Solicita adicionar un torque al cuerpo, 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.

Tabla 8. Caso de Uso expandido: Adicionar material.

Caso de uso	
CU-4	Adicionar material.
Propósito	Adicionar un material al CR
Actores	Programador
Resumen: El caso de uso se inicia cuando el programador solicita crear un nuevo tipo de material introduciendo los coeficientes de fricción estática, cinética, restitución y cuando solicita adicionarle un material a un objeto. El caso de uso finaliza cuando es adicionado el material al cuerpo.	
Referencias	RF-6.1, 6.2
Acción del actor	Respuesta del sistema
1- Se solicita crear un nuevo tipo	2- Se crea el nuevo material.

de material introduciendo el nombre y los coeficientes de fricción.	
3- Se solicita crear una interacción entre dos materiales existentes, introduciendo los nombres de los dos materiales y los coeficientes de fricción correspondiente al tipo de interacción que se desea crear entre estos dos materiales	4- Se crea la interacción entre los dos materiales.
Precondiciones	Tiene que existir al menos un cuerpo en la escena para poder adicionarle un tipo de material.
Poscondiciones	Queda el cuerpo con su nuevo tipo de material.

Tabla 9. Caso de Uso expandido: Crear unión entre dos cuerpos.

Caso de uso	
CU-5	Crear unión entre dos cuerpos.
Propósito	Crear una unión entre dos cuerpos para restringir el movimiento de estos.
Actores	Programador
Resumen: El caso de uso se inicia cuando el programador solicita crear una unión.	
Referencias	RF-9 a), b), c).
Acción del actor	Respuesta del sistema
1- Se solicita crear una unión entre dos cuerpos introduciendo: • Punto de unión entre los cuerpos. • Ángulo de movimiento entre los cuerpos. • Vector el eje de rotación de	2- Se crea la unión entre los dos cuerpos.

la unión de los cuerpos.	
Precondiciones	Tienen que existir dos objetos en la escena para establecer la unión.
Poscondiciones	Quedan los objetos unidos mediante una unión.

Tabla 10. Caso de Uso expandido: Actualizar estado del CR.

Caso de uso	
CU-6	Actualizar estado del CR.
Propósito	Actualizar el estado del CR en cada iteración de la simulación.
Actores	Programador
Resumen: El caso de uso se inicia cada vez que se realice un paso de la simulación. Actualiza estado del CR (posición y orientación). Una vez iniciado se calculan las fuerzas y torques resultantes actuando sobre el CR. El caso de uso finaliza cuando se actualiza el estado del CR.	
Referencias	RF-5
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, se analizan las fuerzas que actúan en el cuerpo y las respuestas a colisiones y se actualiza la nueva posición y orientación del cuerpo teniendo en cuenta los factores planteados anteriormente.
Precondiciones	Tiene que existir al menos un objeto en la escena para actualizar su estado.
Poscondiciones	Queda actualizado el estado físico del objeto (posición y orientación) según leyes físicas.

Conclusiones del capítulo:

En el presente capítulo se definió el sistema propuesto y quedan creadas las bases técnicas por las que se regirá. Quedaron establecidos los requisitos funcionales de éste componente para la física que permitirá al usuario final obtener los resultados esperados por el cliente.

Capítulo 3: Diseño 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 componente físico, que serían diagramas de clases y diagramas de paquetes de la realización de los casos de uso. 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 aclaran cuestiones importantes que permiten la comprensión de los diagramas de clases paquetes:

Se hace uso en el diseño de las clases de los estándares de codificación para el lenguaje c++ utilizado en el centro VERTEX.

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 componente 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 “Component_Physics” es el que agrupa las clases desarrolladas donde se encuentran implementadas las funcionalidades para integrar física a los LV. El paquete “Arquitectura de los Laboratorios Virtuales” como su nombre lo indica es una representación donde se muestra la capa de soporte que contiene los componentes dinámicos que sirven de apoyo a la realización de un ejercicio determinado. El paquete “PAL” es una representación de la capa de abstracción que será de utilidad del componente dinámico para incluir física.

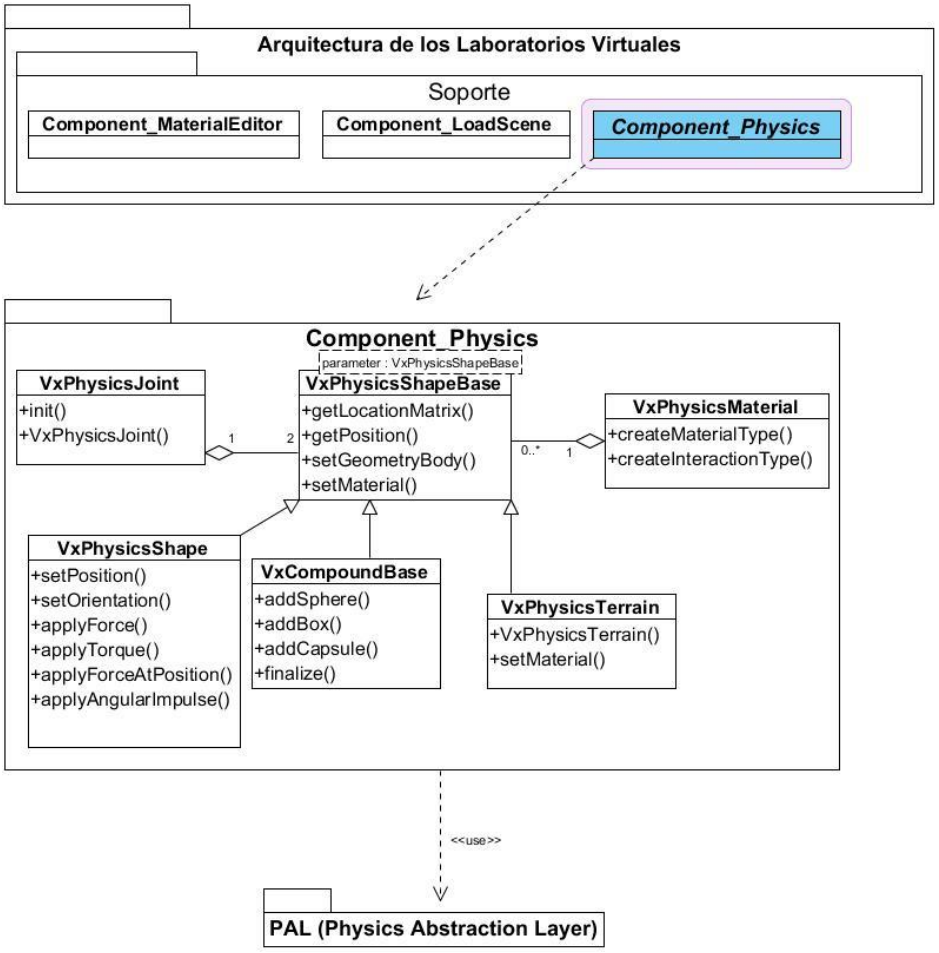


Figura 26. Diagrama de paquetes de clases de diseño.

3.1.2 Diagramas de Clases del Diseño

Las clases necesarias para el funcionamiento del sistema a desarrollar tienen responsabilidades estrechamente relacionadas, pero debido a la extensión del diagrama de clases del diseño se agrupan las clases por funcionalidades.

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

Propiedades del mundo físico.
Aplicar Fuerzas a un cuerpo.
Actualizar estado del cuerpo.
Aplicar material a un cuerpo.
Crear uniones entre cuerpos.

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

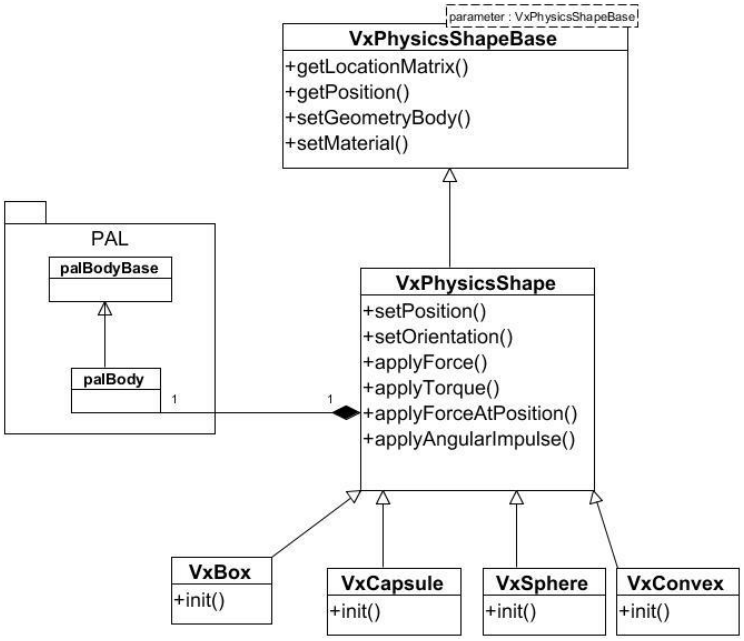


Figura 27. Diagrama de clases de diseño: Inicializar Cuerpo Rígido y actualizar su posición y orientación.

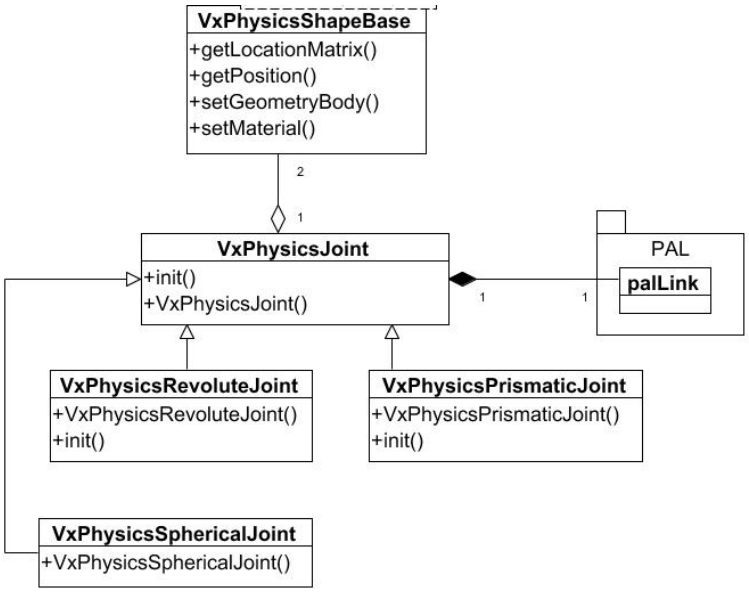


Figura 28. Diagrama de clases de diseño: Crear unión entre cuerpos.

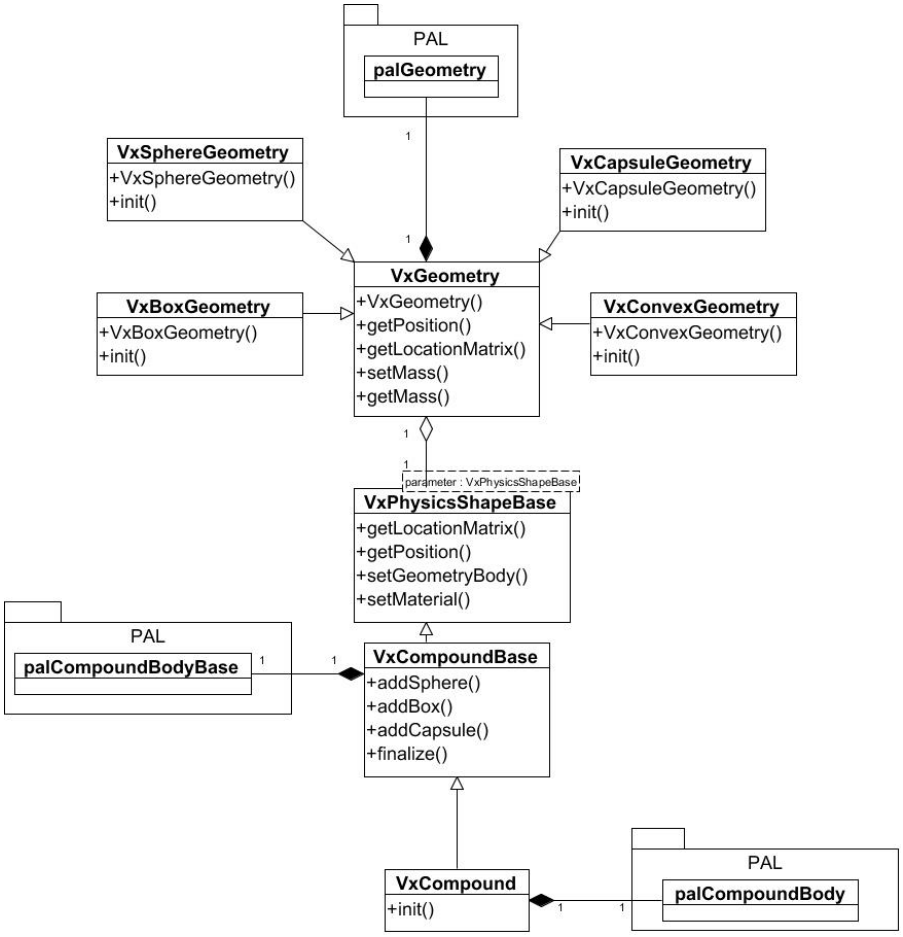


Figura 29. Diagrama de clases de diseño: Crear cuerpos compuestos.

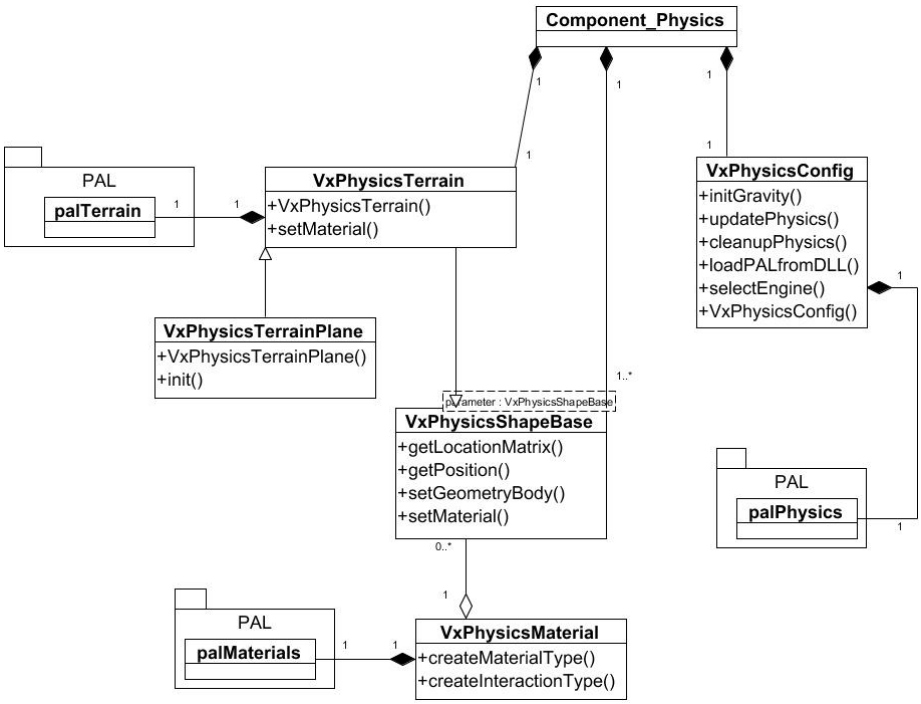


Figura 30. Diagrama de clases de diseño: Crear terreno, aplicar material a un cuerpo y configurar la física del mundo.

3.2 Patrones de diseño utilizados

Un patrón de diseño es una descripción de clases y objetos que se comunican entre sí, adaptados para resolver un problema general de diseño en un contexto particular.

En el diseño del componente de física se tuvo en cuenta principalmente el patrón estructural *Adapter* siendo muy eficaz y sencillo. Convierte la interfaz de una clase en otra distinta, que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatible.

A continuación se muestra el uso del patrón *Adapter* en las clases del componente físico.

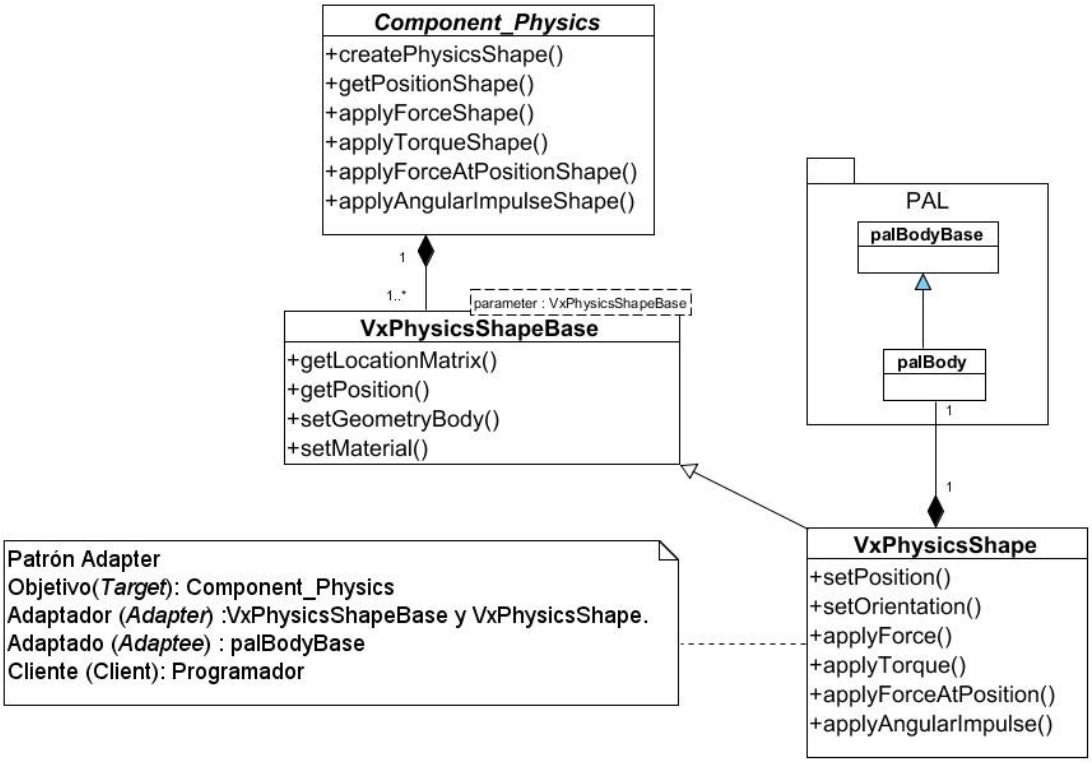


Figura 31. Diagrama de clases de diseño: Aplicación del patrón Adapter.

También se tuvo en cuenta los patrones Experto y Creador. El primero plantea que siempre se debe asignar una responsabilidad al experto en información, es decir, la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. 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.

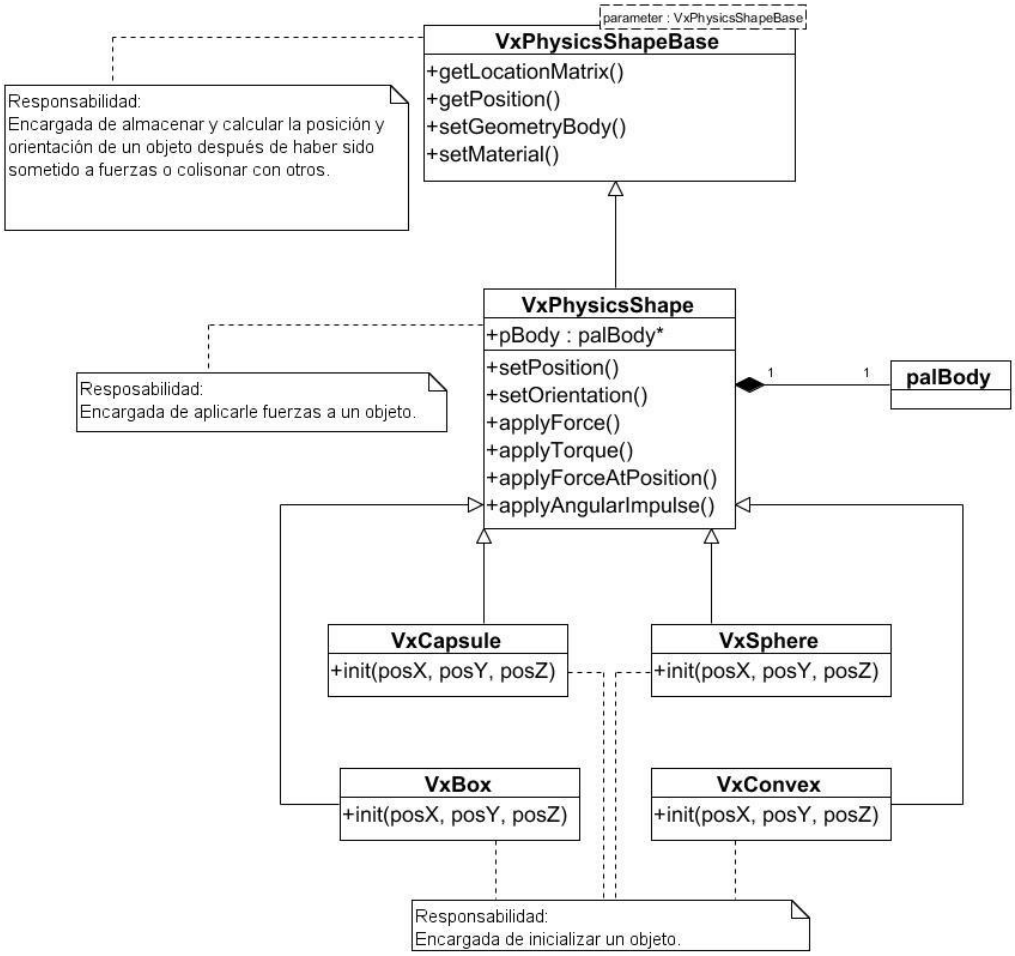


Figura 32. Diagrama de clases de diseño: Aplicación del patrón Experto y Creador.

El patrón Experto sugiere que la clase **VxPhysicsShapeBase** es el experto en información por lo tanto debe asumir la responsabilidad de conocer la posición y orientación de un cuerpo en la simulación. El patrón creador sugiere que la clase **VxPhysicsShape** es idónea para asumir la responsabilidad de crear un puntero de **palBody**.

Consideraciones parciales

Al concluir este capítulo se han establecidos las bases para comenzar la implementación del sistema. A través de los diagramas de paquetes, de clases y los patrones de diseño se han detallado las responsabilidades de cada una de las clases que serán utilizadas. Una vez terminado el diseño se pasa a la implementación en el siguiente capítulo.

Capítulo 4: Implementación y prueba

Introducción

En el presente capítulo se hace referencia a la etapa de codificación y pruebas, al concluirlo se tendrá un componente físico implementado y probado.

4.1 Pruebas y resultados

Las pruebas del presente trabajo, se dividen en dos grupos. El primer grupo se centra en el componente físico y su funcionalidad mientras que el segundo grupo de pruebas se centra en los motores físicos y su desempeño.

A continuación se muestran códigos de ejemplos utilizando el componente desarrollado.

El código que se muestra a continuación es un caso básico en el cual se inicializa y crea un mundo virtual de simulación física, se selecciona el motor físico ODE en su versión 0.11.1 y se crea un objeto (caja). El mismo código puede ser reutilizado por el programador en distintos motores físicos, únicamente cambiando la tercera línea de código (el nombre del motor físico).

```
VxPhysicsConfig * conf;
conf->loadPALfromDLL("Dir donde se encuentran las bibliotecas");
conf->selectEngine("ODE");
conf->initGravity();

VxPhysicsTerrainPlane * vxTerrainPlane = new VxPhysicsTerrainPlane();

vxTerrainPlane->init(0,0,0,25.0f);

VxBox * vbox = new VxBox();

vbox->init(x,y,z,dimX,dimY,dimZ,mass);
vbox->setMaterial("Normal");

conf->updatePhysics(0.01f);
conf->cleanupPhysics(); // dejar de utilizar el motor físico ODE
```

En el código que se muestra a continuación se crea un mundo virtual semejante al anterior ejemplo utilizando el mismo motor físico inicialmente, luego se utiliza el método **cleanupPhysics()** el cual permite eliminar todas las configuraciones y objetos creados en el mundo físico, libera toda la memoria utilizada por el motor físico. Cualquier uso de otras referencias a objetos de la física queda indefinido, necesitando inicializar otro motor de simulación física. En el ejemplo se inicializa la versión de *Bullet* 2.82 y se vuelve a inicializar el mundo virtual físico, luego se crean dos objetos para interactuar en escena.

```
VxPhysicsConfig * conf;
conf->loadPALfromDLL("Dir donde se encuentran las bibliotecas");
```

```
conf->selectEngine("ODE");
conf->initGravity();

VxPhysicsTerrainPlane * vxTerrainPlane = new VxPhysicsTerrainPlane();

vxTerrainPlane->init(0,0,0,25.0f);

VxBox * vbox = new VxBox();

vbox->init(x,y,z,dimX,dimY,dimZ,mass);
vbox->setMaterial("Normal");

conf->updatePhysics(0.01f);
conf->cleanupPhysics(); // dejar de utilizar el motor físico ODE

// Configurar denuevo la fisica del mundo e inicializar los objetos

VxPhysicsConfig * conf;
conf->loadPALfromDLL("Dir donde se encuentran las bibliotecas del
motor físico a utilizar");
conf->selectEngine("Bullet");
conf->initGravity();

VxPhysicsTerrainPlane * vxTerrainPlane = new VxPhysicsTerrainPlane();

vxTerrainPlane->init(0,0,0,25.0f);

VxBox * vbox = new VxBox();

vbox->init(x,y,z,dimX,dimY,dimZ,mass);
vbox->setMaterial("Normal");

conf->updatePhysics(0.01f);
conf->cleanupPhysics();
```

Estos mismos conceptos se repiten en casos de prueba más complejos, demostrando que el componente implementado presenta facilidad de uso y brinda la posibilidad de utilizar los distintos motores físicos con un mínimo esfuerzo por parte del programador, permite el uso de varios motores físicos en una misma simulación (aclarar que no quiere decir que dos motores de simulación física se utilicen a la misma vez en tiempo real).

Sobre dos motores físicos se realiza una prueba; que compara los resultados de la simulación con el valor exacto de esa simulación calculado mediante las fórmulas físicas correspondientes para resolver esa situación en la vida real. Esta prueba tiene como objetivo tener una medida de que tan real es la simulación realizada utilizando el componente desarrollado.

La simulación se realiza en una escena sencilla para lograr que sea lo más cercana posible a la realidad sin que interfieran fenómenos de rendimiento en el resultado final.

Se aprovecha la facilidad de uso que brinda el componente para intercambiar motores en una aplicación, los dos motores físicos utilizados serán ODE 0.11.1 y *Bullet* 2.82.

Estas dos bibliotecas reúnen las mejores características en general según la comparación que se realizó en el capítulo 2 y se adaptan a las necesidades del proyecto.

4.1.1 Prueba de comportamiento: cuerpo en caída libre

Se realizaron una cantidad considerable de pruebas en ambos motores para distintos casos con solución conocida, para poder evaluar su comportamiento y precisión: caída libre de un cuerpo.

Los motores físicos pueden ser configurados para obtener mayor o menor grado de precisión, seleccionando el paso de tiempo de actualización de la simulación. A menor número del paso del tiempo, más precisa será la simulación y viceversa.

El siguiente ejercicio se elaboró por un profesor especialista en física de la facultad y será utilizado para realizar la prueba.

Yanet deja caer desde el 2do piso un celular para que su hermana Aida lo reciba, la ventana desde donde se deja caer está a una altura de 8,52 m hasta las manos de Aida. Determina que tiempo transcurre para que el celular llegue a las manos de Aida.

Información conocida:

En forma de variable solamente hay un dato explícito 8,52 m.

El resto de información debe ser extraída de acuerdo al entendimiento de los principios de la caída libre:

La distancia o altura (y) es 8,52 m.

La velocidad inicial (V_0) puede deducirse como 0 m/s.

La aceleración de la gravedad (g) se puede seleccionar como 9,8 m/s^2.

En la caída libre no se tiene en cuenta la resistencia del aire.

Dos cuerpos caerán sobre la superficie de la tierra a la misma velocidad, solo si tienen la misma masa y el mismo volumen. Ahora bien si la distancia de caída es muy pequeña, la aceleración de los cuerpos (g) es muy pequeña y aunque tuvieran distinta masa podrían caer casi a la vez. Se desprecia la masa en este caso.

Datos:

V_0= 0,0 m/s

y = 8,52 m

a = g = 9,8 m/s^2

Encontrar

t = ?

Desarrollo

Ecuación de la Dinámica en caída libre para determinar la altura.

$$y = y_0 + V_0 t - \frac{1}{2} g t^2$$

$$y = 8.52m + (0m/s^2)(t) - \frac{1}{2}(9.8\,m/s^2)(t^2)$$

$$t^2 = \frac{-8.52m}{-4.9\,0m/s^2}$$

$t^2 = 1.739s^2$

$t = 1.3s$

Para recrear el ejercicio anteriormente planteado se realizó una simulación utilizando el componente dinámico en un ambiente tridimensional para medir el tiempo que insume el motor físico en simular un objeto que cae en un plano estático desde 8.52m de altura. La gravedad del mundo es 9,81 m/s^2 y el paso del tiempo utilizado varía en cada prueba realizada. Los resultados de las pruebas se detallan en las tablas siguientes.

Tabla 11. Pruebas utilizando el motor físico ODE.

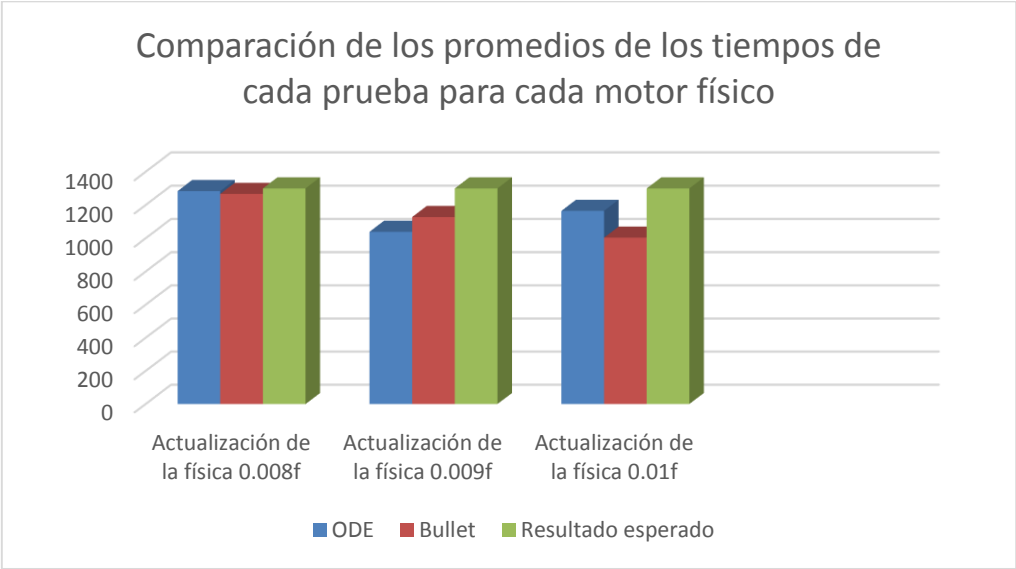
Cantidad de iteraciones por prueba	Paso de actualización de la física	Tamaño del objeto en X	Tamaño del objeto en Y	Tamaño del objeto en Z	Altura del objeto en en Y	Gravedad 9.81 m/s^2	Masa	Tiempo en milisegundos de la simulación
1	0.008f	1m	1m	1m	8.52 m		1kg	1391
2								1282
3								1297
4								1282
5								1219
6								1375
7								1250
8								1328
9								1204
10								1266
Tiempo promedio								1284
1	0.009f	1m	1m	1m	8.52 m		1kg	1031
2								1015
3								1109

4								1032
5								1016
6								1031
7								1102
8								1015
9								1017
10								1030
Tiempo promedio								1039
1	0.01f	1m	1m	1m	8.52 m		1kg	1141
2								1140
3								1266
4								1140
5								1140
6								1219
7								1157
8								1141
9								1172
10								1141
Tiempo promedio								1165

Tabla 12. Pruebas utilizando el motor físico Bullet.

Cantidad de iteraciones por prueba	Paso de actualización de la física	Tamaño del objeto en X	Tamaño del objeto en Y	Tamaño del objeto en Z	Altura del objeto en en Y	Gravedad 9.81 m/s^2	Masa	Tiempo en milisegundos de la simulación
1	0.008f	1m	1m	1m	8.52 m		1kg	1265
2								1281
3								1281
4								1266
5								1203
6								1281
7								1281
8								1297
9								1281
10								1234
Tiempo promedio								1267
1	0.009f	1m	1m	1m	8.52 m		1kg	1125
2								1140
3								1094
4								1141
5								1141
6								1172
7								1078
8								1125

9								1125
10								1140
Tiempo promedio								1128
1	0.01f	1m	1m	1m	8.52 m		1kg	1015
2								969
3								968
4								984
5								1016
6								1031
7								1032
8								1047
9								969
10								1015
Tiempo promedio								1004



Gráfica 1. Comparación de los tiempos promedios de pruebas de cada motor de simulación física.

4.1.2 Resultado de las pruebas

Luego de realizar la comparación se puede apreciar que el tiempo promedio para la actualización de la física con un paso de 0.008f en ambos motores es muy cercano al resultado esperado, siendo ODE más exacto. Para la actualización de la física con un paso de 0.009f se nota como el tiempo promedio de ambos motores disminuye alejándose del resultado esperado, en este caso *Bullet* tiene mejores resultados de precisión. Para un paso de actualización de la física de 0.01f se aprecia como ODE se acerca al resultado esperado y *Bullet* se aleja un poco más. Se puede notar como a

mediada que el paso de la simulación crece el motor físico busca más rapidez en la simulación siendo menos preciso. Esta es una característica fundamental de estos dos motores físicos, precisan la velocidad de la simulación por encima de la exactitud física.

Se puede concluir que entre estos dos motores ODE es el que mejor exactitud física posee, buscando *Bullet* mucha más velocidad en la simulación lo que no quiere decir que no se pueda buscar más precisión solo hay que variar el tiempo del paso de la simulación hasta lograr los resultados deseados.

Conclusiones del capítulo

En el presente capítulo fueron abordadas diferentes aristas orientadas a la codificación y realización de las pruebas definidas para la validación de la solución implementada. Luego de haber realizado las pruebas correspondientes, se puede afirmar que se cuenta con un software que cumple con los requisitos planteados.

Conclusiones Generales

El componente desarrollado integra la dinámica de CR a la arquitectura para laboratorios virtuales, lo que permite aumentar el realismo de las escenas simuladas y ampliar el rango de aplicaciones que se pueden desarrollar.

Recomendaciones

- Utilizar el componente dinámico para integrar física a los laboratorios virtuales lo que ampliaría el rango de aplicaciones que se pueden desarrollar.
- Incluir la simulación de la dinámica de los cuerpos blandos abarcar un mayor número de campos de aplicación.

Referencias bibliográficas

1. *INCORPORACIÓN DE COMPORTAMIENTO FÍSICO EN MOTORES.* **Cardona, Alberto, Storti, Mario y Zuppa, Carlos.**
2. **Shannon, Robert y Johannes, James D.** *«Systems simulation: the art and science». IEEE Transactions on Systems, Man and Cybernetics.*
3. **Doménech Osete, Manel R.** Arquitectura para la gestión de modelos articulados. [En línea] http://mural.uv.es/mado/odex/docs/memoria/html/memoria.html.
4. **Cortadella Fortuny, Jordi .** Departamento de lenguajes y sistemas informáticos. [En línea] www.lsi.upc.edu/~virtual/SGI/guions/.
5. **Stepien, Jakub.** *Physics-Based Animation of Articulated Rigid Body Systems for Virtual Environments.* Silesian University of Technology Faculty of Automatic Control, Electronics and Computer Science Institute of Informatics : s.n., 2013.
6. *Real-time Animation of Complex Virtual Cloth with Physical.Plausibility and Numerical Stability. Teleoperators & Virtual Environments.* **Young-Min, Kang.** 2004.
7. ***Beer, F.P., 1998, "Mecánica vectorial para ingenieros", 6ta Ed, MacGrawHill, 1998. 4ta Ed. pueblo y eduación, La Habana, Cuba.***
8. Sears, F. W., Zemansky,M. W. and Young,H. D., 1998, "Física Universitaria", 9na Edición , Addison-Wesley Longman.
9. *Diagonalización del Tensor de Inercia para mejorar la eficiencia en la Simulación de Cuerpos Rígidos.* Palomino Ramírez, Luis y Rudomin Goldberg, Isaac. Computacion Visual 1997.
10. *Deformación Plástica. Mecánica de sólidos deformables.* Rosario, L.M.
11. *Taller Multimedia I.Capítulo 9: Animación.* Barreiro, L.N.P. México : s.n., 2004.
12. *Introdución a la dinámica de los cuerpos rígidos.* Rico Martínez, José María . Departamento de Ingeniería Mecánica División de Ingenierías, Campus Irapuato-Salamanca Universidad de Guanajuato. Salamanca México : s.n., 4 de junio del 2013.
13. González Morcillo, Carlos, y otros. *Desarrollo de Video Juegos: Programación Gráfica.* s.l. : Bubok(Edición Física). ISBN 97-884-686-1058-0.
14. *Simulación de la Dinámica de Cuerpos Rígidos en Tiempo Real .* Curello, Pablo Andrés. Buenos Aires : s.n.
15. *DINÁMICA DE UN CUERPO RÍGIDO.* ARENAS GAVIRIA, BERNARDO. Universidad de Antioquia, Instituto de Física : s.n., 2012.
16. Dávila Baz, Juan Antonio y Pajón Permuy, Javier. *MECÁNICA GENERAL.* Universidad de HUELVA : s.n., 1999. ISBPN.
17. *Lecciones de Física (4 volúmenes) (en español). Monytex.* Ortega, Manuel R. abril 19, 2010, pág. 24. ISBN 84-404-4290-4,ISBN 84-398-9218-7, ISBN 84-398-9219-5, ISBN 84-604-4445-7.
18. *Rigid Body Simulation.* Jenkins López, David B , Freitas, Álvaro del Monte y Montenegro Montes, Manuel . julio 3, 2006.
19. Ortega Girón, Manuel R. . *Lecciones de Física, Mecánica 2.* Departamento de Física Aplicada. Universidad de Córdoba : s.n., 2006. ISBN 84-398-9218-7.
20. Moya Fernández, Francisco , y otros. *Desarrollo de Video Juegos, Técnicas Avanzadas.* ISBN 978-84-686-1059-7.
21. *Rigid Body Collision Response.* Kavan, Ladislav . Faculty of Mathematics and Physics, Charles University, Prague / Czech Republic : s.n.
22. San Román, Dr. Segundo Esteban , y otros. *MOTOR DE FÍSICA MULTIFUNCIONAL.* FACULTAD DE INFORMÁTICA, UNIVERSIDAD COMPLUTENSE DE MADRID, PROYECTO DE SISTEMAS INFORMÁTICOS : s.n., 2010.
23. *Open dynamics engine v0.8 user guide.* Rusell, Smith. Mayo 2007.
24. Smith, Russell . Open Dynamics Engine . [En línea] 2008. http://www.ode.org/.

25. Jerez, Julio y Suero, Alain . Newton Game Dynamics. [En línea] 2008. http://newtondynamics.com/forum/newton.php.
26. Nvidia PhysX. [En línea] 2014. http://www.nvidia.com/object/nvidia_physx.html..
27. Havok Physics. [En línea] 2014. http://www.havok.com/.
28. Bullet Physics Library. [En línea] http://bulletphysics.org/wordpress/.
29. Boeing, Adrian . Physics Abstraction Layer. [En línea] http://www.adrianboeing.com/pal/.
30. *Architectural Blueprints—The "4+1" View Model of Software Architecture.* Kruchten, Philippe . s.l. : IEEE Software, 1995.
31. *Recommended Practice for Architectural Description of Software-Intensive Systems.* Hilliard, Rich . s.l. : IEEE-Std-1471-2000.
32. *Informe de la reunión de expertos sobre laboratorios virtuales. Instituto Internacional de Física Teoríca y Aplicada(IITAP).* Ames, Iowa y James P. Vary. París : CII-2000/WS/1, 2000.
33. Merzeau Martínez, Alejandro D. *Arquitectura de software para los Laboratorios Virtuales.* La Habana : s.n., 2012.
34. *Ampliación de la Informática Gráfica Tema 5 Animación 3D 2007.* Universidad de Valencia : s.n.
35. Microsoft. MSDN Library. Quaternion Structure. [En línea] 2007. http://msdn2.microsoft.com/en-us/library/microsoft.windowsmobile.directx.quaternion.aspx.
36. *Entorno de Simulación Robótico.* Figueroa, Anthony, y otros. Instituto de Computación, Facultad de Ingeniería - Universidad de la República, Montevideo - Uruguay : s.n., 2008.
37. *Patrones de diseño aplicados al desarrollo de objetos digitales educativos (ODE).* Aedo, Ignacio, y otros. s.l. : Instituto de Tecnologías Educativas, 2011. ISBN eBook: 978-84-369-5251-3.
38. *Design Patterns: Elements of reusable object-oriented software. s.l. : Addison-Wesley.* Gamma, Erich, y otros. 1995.

Glosario de términos

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.

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

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.

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

Escena: cada una de las partes de una obra dramática, una película o una animación. Representa una determinada situación con los mismos personajes u objetos.

Propiedades físicas: propiedades de los objetos del mundo real que se simularán con los objetos virtuales a través de la manera de ejecutar sus tareas, por ejemplo, la masa.

Realidad Virtual: simulación generada por computadora de imágenes o ambientes tridimensionales interactivos con cierto grado de realismo físico o visual.

Vector: cantidad que expresa magnitud y dirección.

Virtual: término utilizado para referirse a algo que no tiene existencia física o real, sólo aparente.