Estado del arte

Entorno de Simulación Robótico

Anthony Figueroa

pgsimrob@fing.edu.uy

Tutor

Gonzalo Tejera

Cotutores

Gustavo Armagno, Facundo Benavides, Serrana Casella

15 de mayo de 2007

Instituto de Computación

Facultad de Ingeniería - Universidad de la República

Montevideo - Uruguay

Resumen

Este documento presenta el estado del arte de algunos de los simuladores de fútbol y sumo de robots existentes en la actualidad y los mecanismos o herramientas utilizadas para llevar a cabo los mismos. Es de vital importancia para comprender mejor el funcionamiento de algunos de estos simuladores realizar un relevamiento de las principales herramientas de simulación física disponibles, ya que se considera que para lograr una simulación confiable, robusta, y en un entorno que sea configurable, es fundamental contar con un motor físico sobre el cual pueda construirse un simulador con estas características. Además se realiza un estudio de los diferentes motores gráficos de uso libre en la actualidad, estudiando la integración de los mismos con diferentes motores físicos. Se realiza un estudio del funcionamiento de diferentes simuladores de fútbol de robots, como "3D Robot Soccer Simulation", "Mis20", "Robocup Soccer Simulator", entre otros.

Índice

Índice de figuras

1. Introducción

El concepto de inteligencia artificial es complejo de definir y existen diferentes enfoques e interpretaciones. Una definición muy difundida fue propuesta por Marvin Minsky, profesor del MIT (Massachusetts Institute of Technology) y es la siguiente: "la inteligencia artificial es la ciencia de construir máquinas que hacen cosas que de ser realizadas por el hombre requieren el uso de inteligencia".

La creación de diferentes tipos de competencias entre robots es un intento por estimular a nivel académico o personal la investigación en el área de inteligencia artificial. Torneos de sumo o fútbol de robots son un ejemplo de dichas competencias. Se logra de esta manera, una aproximación a diversas ramas de la inteligencia artificial a través de la creación de robots que participen en estos torneos. Por ejemplo, para la creación de un equipo competitivo de fútbol de robots se debe hacer frente a problemas muy diversos, como son el planeamiento de trayectorias (path planning), procesamiento de imágenes, aprendizaje automático, cooperación entre agentes y procesamiento en tiempo real, entre otros.

La utilización de simuladores es vital para concentrar mayores esfuerzos en los problemas que aparecen en la programación del comportamiento de los robots y abstraerse de problemas que normalmente aparecen cuando se trabaja sobre modelos reales, como la propia construcción de los robots, así como también agotamiento de baterías, problemas de comunicación, etc. Centrando los esfuerzos en la optimización del comportamiento utilizando simuladores, se puede lograr la obtención de resultados interesantes, y utilizar un simulador con las características apropiadas es importante para ello.

En el contexto de este proyecto se pretende crear un entorno de simulación en el cual sea posible realizar pruebas de estrategias de control de robots, tanto para su utilización en equipos de fútbol como para competencias de sumo de robots. Por lo tanto, dicho entorno debe ser lo suficientemente flexible y genérico para lograr modelar ambas realidades.

Este documento presenta el estado del arte de algunos simuladores de fútbol y sumo de robots existentes en la actualidad. Asimismo para comprender mejor el funcionamiento de algunos de estos, es de vital importancia realizar un relevamiento de las principales herramientas de simulación física existentes, ya que se considera que para lograr una simulación confiable, robusta, y en un entorno genérico, es fundamental contar con un motor físico sobre el cual sea posible construir un simulador con estas características. Además se realiza un estudio algunos motores gráficos de uso libre, estudiando su integración con diferentes motores físicos.

En las diferentes secciones de este documento se cubren temas referentes a motores físicos, motores gráficos y simuladores existentes. Es importante catalogar las ventajas y desventajas de cada uno de estos, para que se posean las herramientas necesarias para tomar buenas decisiones y de esta manera crear el contexto necesario para desarrollar un simulador genérico de fútbol y sumo de robots .

En el capítulo 2 se evalúan diferentes motores físicos disponibles, tanto los de

código abierto como cerrado, tanto los comerciales como los de uso libre. Éste es un tema muy importante, y se pone énfasis en características del simulador como robustez, precisión y estabilidad, ya que éstas son características importantes para lograr una simulación fiel a la realidad.

En el capítulo 3 se evalúan diferentes motores gráficos. Estudiando su integración con motores físicos.

En el capítulo 4 se describirá y evaluará la factibilidad del uso de una herramienta para crear simuladores llamada "Phi".

En el capítulo 5 se describirán algunos simuladores de fútbol de robots disponibles en la actualidad. Se pondrá énfasis en estudiar su arquitectura interna si la misma está disponible y prestaciones, además se evaluarán los métodos utilizados para resolver la simulación física y su interfaz gráfica.

2. Motores Físicos

2.1. Introducción

El modelo físico sobre el cual se debe basar un simulador de fútbol o sumo de robots debe ser tan fiel a la realidad como sea posible. Una alternativa es utilizar un motor físico para implementar dicho modelo. Existen diversas características deseables con las cuales debería contar un motor físico utilizado en el marco de este proyecto. La precisión en los cálculos matemáticos y la estabilidad de los mismos es muy importante, ya que se desea que el modelo físico sea tan fiel a la realidad como sea posible. Además, el simulador a construirse debe ser genérico y debe poder simular realidades tan diversas como el sumo de robots o distintas categorías de fútbol de robots, por lo que es deseable que sea posible modelar esta realidad en el motor físico.

Por otra parte, otra característica importante es que exista la posibilidad de controlar de manera flexible el control del paso de simulación, para lograr entre otras cosas acelerar el paso del tiempo. También sería interesante que el motor físico brindara algún tipo de funcionalidad para almacenar el estado de la simulación o incluso simulaciones enteras. Esto facilitaría la tarea de grabación de simulaciones, que podría tener como aplicación almacenar partidas enteras de fútbol o sumo de robots para luego visualizarlas con más detenimiento.

La presencia de los siguientes elementos son denominadores comunes en todos los motores físicos.

2.1.1. Cuerpos Rígidos

Los cuerpos rígidos son la piedra angular de la simulación. Éstos se conectan mediante empalmes. La forma de los mismos permanece incambiada a lo largo de la simulación, siendo esta forma determinante para resolver como colisionan los cuerpos.

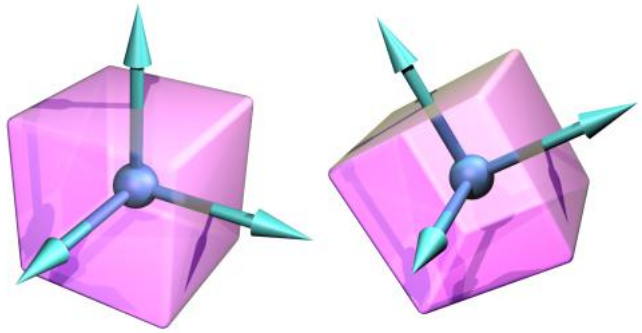

Figura 1: Cuerpos rígidos y sus ejes (extraído de [Smi07]).

2.1.2. Empalmes

Los empalmes se utilizan para conectar de alguna manera dos objetos en la simulación. Estos empalmes están sujetos a restricciones, que determina la manera en que están habilitados a moverse 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.

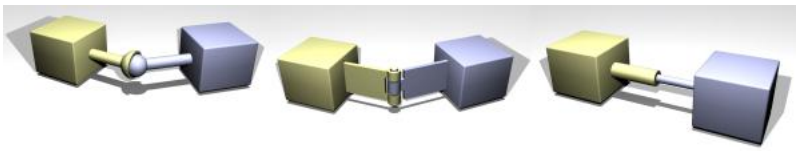

Figura 2: Empalmes (extraído de [Smi07]).

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.

2.2. ODE

La librería ODE[1] es gratuita, de código abierto y ampliamente utilizada. Según [Smi07] puede utilizarse para usos muy variados, es relativamente rápida, flexible y robusta y además presenta la propiedad de detección y manejo de colisiones. Es buena para simular estructuras de cuerpos rígidos articulados. Para crear estructuras de esta naturaleza se conectan cuerpos rígidos de diferentes tamaños y formas mediante el uso de empalmes de diferentes características y con diferentes grados de libertad. Ejemplificando, se pueden modelar vehículos en los cuales las ruedas están conectadas al chasis mediante empalmes, o criaturas con patas en las cuales dichas patas están conectadas al cuerpo.

ODE está diseñado para ser utilizado en simulaciones interactivas o en tiempo real, en particular simular objetos que se mueven en un contexto en el cual la realidad virtual cambia. Además el usuario tiene la libertad de cambiar muchos aspectos de dicha simulación aún luego de haber comenzado la misma.

Un punto importante de los motores físicos a tener en cuenta es su estabilidad, o sea, que lo errores de cálculo no crezcan de forma descontrolada. Según su documentación, ODE tiene como característica un muy buen manejo de estos errores, evitando que el sistema se vuelva matemáticamente inestable sin una

[1] Open Dynamics Engine

razón aparente, logrando esto a partir del hecho que se prioriza la estabilidad y la velocidad de la simulación por sobre la precisión física.

A continuación se cubrirán algunos conceptos importantes cuando se desea utilizar este motor físico.

2.2.1. Cuerpos Rígidos

Los cuerpos rígidos presentan diferentes propiedades que vale la pena destacar:

- Vector posición: Coordenadas (x,y,z) del centro de masa del cuerpo.
- Vector Velocidad Lineal: Representa la velocidad lineal (vx,vy,vz) del centro de masa del cuerpo.
- Orientación del cuerpo, representada por una cuaterna (qs,qx,qy,qz) o una matriz de rotación de dimensiones 3x3. En la cuaterna los valores de qx,qy y qz reflejan el ángulo que forma el cuerpo con respecto a cada uno de los ejes de coordenadas. El valor de qs corresponde a un valor de normalización de los demás componentes de la cuaterna, reescalando los mismos.
- Vector Velocidad Angular: Representa la velocidad angular (wx,wy,wz) del centro de masa.
- Masa del cuerpo.
- Posición del centro de masa con respecto a un punto de referencia.
- Matriz de Inercia del cuerpo: Describe como la masa del cuerpo está distribuida alrededor del centro de masa.

Cabe destacar que los cuerpos pueden ser deshabilitados a lo largo de la ejecución de la simulación, eliminando todo tipo de actualización y control de colisiones sobre los mismos, de manera que la simulación sea más liviana y rápida, ahorrando recursos computacionales.

2.2.2. Empalmes

Los tipos de empalmes presentes en esta distribución de ODE son:

- 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.

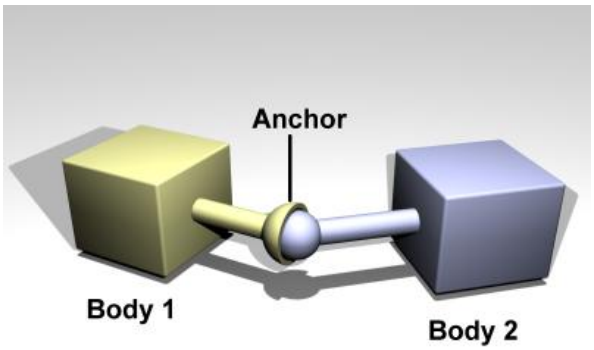

Figura 3: Empalme *ball and socket* (extraído de [Smi07]).

- 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.

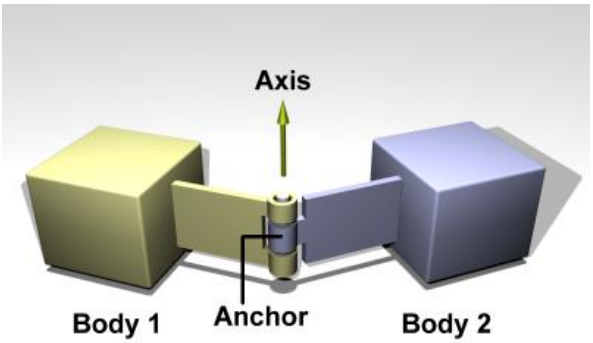

Figura 4: Empalme *hinge* (extraído de [Smi07]).

- 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.

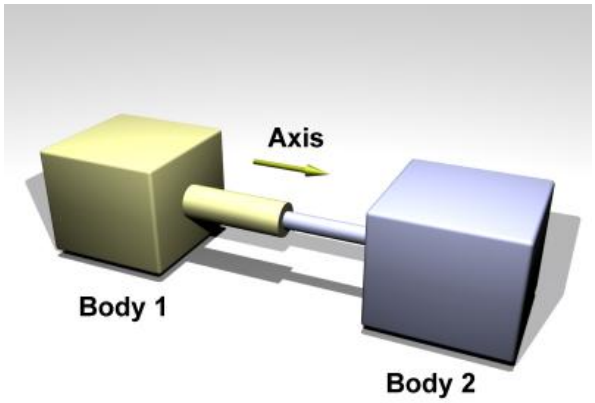

Figura 5: Empalme *slider* (extraído de [Smi07]).

- Universal: El comportamiento de este empalme es análogo al de tipo "Ball and socket" con la diferencia que presenta un grado de libertad más que dicho empalme. Dado el eje 1 en el cuerpo 1 y el eje 2 en el cuerpo 2, que es perpendicular al eje 1, este empalme mantiene estos ejes perpendiculares. En otras palabras, la rotación de ambos cuerpos con respecto a la dirección perpendicular a ambos ejes, va a ser igual. ([Smi07])

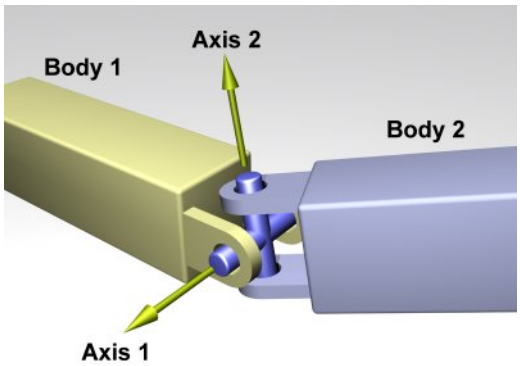

Figura 6: Empalme *universal* (extraído de [Smi07]).

- Hinge-2: Este empalme consta de dos empalmes de tipo "Hinge" conectados en serie, con diferentes ejes.

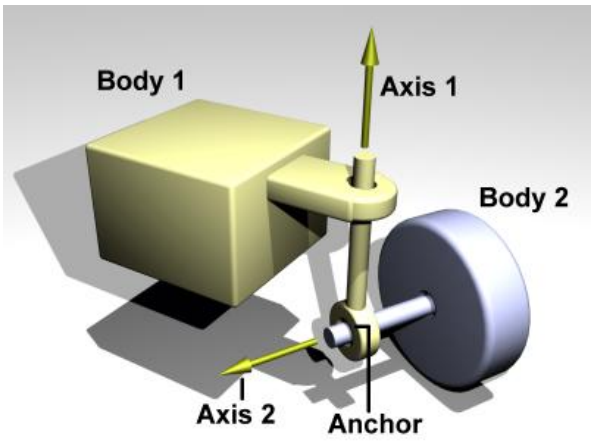

Figura 7: Empalme *hinge-2* (extraído de [Smi07]).

- Fixed: Este empalme mantiene fijas las posiciones relativas entre dos cuerpos. Si dos cuerpos están unidos mediante este empalme, mover uno de ellos resultaría en un movimiento análogo del otro cuerpo. Según [Smi07] no hay buenas razones para utilizar este empalme, excepto para depuración. Si se desea modelar dos cuerpos pegados, lo mejor sería modelarlo como un cuerpo solo.
- Contact: Este empalme previene que dos cuerpos se atraviesen cuando entran en contacto en la simulación. Generalmente es creado en respuesta a la detección de una colisión y desaparece luego de un paso en la simulación.

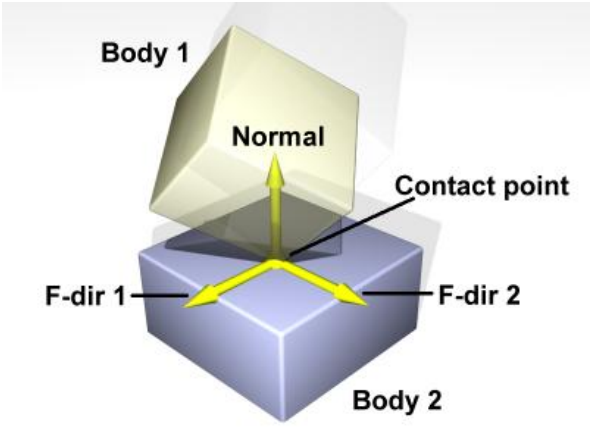

Figura 8: Empalme *contact* (extraído de [Smi07]).

- Motor Angular: Este empalme permite controlar la velocidad angular relativa entre dos cuerpos. Es de gran utilidad si utilizada en combinación del empalme "Ball and Socket", pero puede utilizarse en cualquier situación en la cual control angular es necesario.

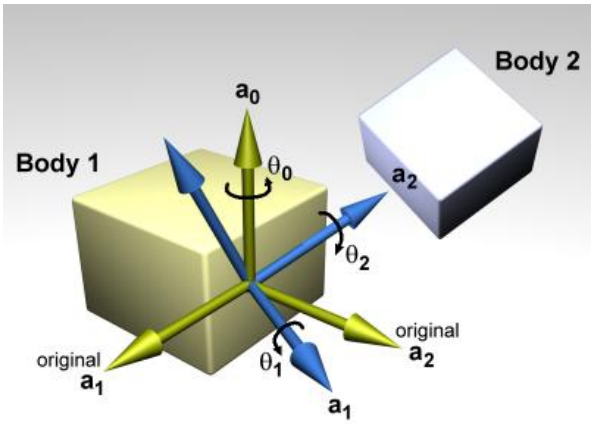

Figura 9: Empalme *motor angular* (extraído de [Smi07]).

2.2.3. Motores y Paradas

Cuando se crea un empalme, no hay nada que lo prevenga de moverse en todo su rango de movimientos, posiblemente infinito. Para lograr establecerle limites a los empalmes se utilizan las paradas (stops), estableciendo de esta manera restricciones que los empalmes deben respetar. Al igual que las paradas, los empalmes pueden tener motores. Estos le aplican una fuerza a un empalme en el sentido de alguno de sus grados de libertad, para que el mismo se mueva a alguna velocidad deseada. Los motores tienen límites de fuerza, por lo que no se puede aplicar mayores torques que un máximo preestablecido a un empalme. Para manipular los motores se utilizan dos parámetros: la velocidad deseada y el torque máximo permitido para lograr tal velocidad.

2.2.4. Colisiones

Cuando el detector de colisiones detecta colisiones, las representa en la forma de un vector de puntos de contacto. Cada punto de contacto contiene la información acerca de sus coordenadas, su vector normal, la profundidad de penetración y que objetos están haciendo contacto. Para manejar las colisiones se debe especificar una función que se ejecuta en cada ciclo de simulación y modela el comportamiento de cada punto de contacto. En estos puntos se puede especificar entre otras cosas la fuerza de rozamiento entre ambos cuerpos.

En el manejo de colisiones de ODE se pueden distinguir dos conceptos importantes:

- Geoms o objetos geométricos: Son los objetos fundamentales del sistema de colisiones. Representa un objeto de forma fija (como una esfera o una caja). Dos Geoms cualesquiera pueden colisionar entre sí, obteniéndose un conjunto de puntos de contacto. Se utilizan para representar cuerpos rígidos en la simulación.

- Spaces: También es un Geom pero tiene la particularidad de agrupar otros Geoms. El objetivo de estos objetos es acelerar el manejo de colisiones de manera sensible, primero agrupando Geoms en Spaces y luego verificando colisiones de Geoms dentro de cada Space, por lo que se puede optar por no verificar las colisiones para todos los pares de Geoms existentes. Además se pueden verificar colisiones entre dos espacios determinados y utilizar jerarquías de espacios.

El sistema de detección de colisiones de ODE puede ser ignorado y de esta manera se lo podría manejar de forma manual, haciéndolo de la manera deseada.

Las primitivas de colisiones actualmente soportadas por el motor físico son:

- Sphere: Representa una esfera

- Box: Representa un prisma

- Capped Cylinder: Representa un cilindro en cuyos extremos tiene medias esferas como terminación

- Plane: Representa un plano geométrico
- Ray: Representa una linea infinitamente fina que empieza en determinada posición y se extiende a lo largo del eje Z.
- Triangular Mesh: Representa una colección arbitraria de triángulos. Se utiliza para representar cuerpos rígidos de cualquier forma, pero utilizar esta primitiva acarrea diversos problemas de estabilidad y velocidad, por lo que en ocasiones puede afectar seriamente la simulación.

2.2.5. Otras características

Algunos parámetros configurables de la simulación que es de gran importancia destacar son:

- ERP (Error Reduction Parameter): Algunas veces por diferentes razones, dos cuerpos unidos por un empalme no cumplen las restricciones que deberían en cuanto a su ubicación, y en cada paso de simulación el empalme realiza una fuerza especial para traer a los cuerpos a una ubicación correcta. Este parámetro determina que porcentaje del error será corregido a cada paso de simulación.
- CFM (Constraint Force Mixing): Se utiliza para suavizar las restricciones tanto en el manejo de empalmes como en las colisiones de cuerpos. Cuanto mayor sea el valor de este parámetro, mayor será la estabilidad de la simulación.
- Gravedad: Se especifica la gravedad del mundo simulado.
- Paso de simulación: Se debe especificar el tamaño del paso de simulación expresado en segundos. Es un parámetro muy importante para lograr una correcta simulación. Al utilizar valores pequeños se obtiene mayor precisión en los cálculos y mayor estabilidad pero en contrapartida se enlentece la velocidad de la ejecución. Utilizar pasos de tiempo de tamaño variable entre una iteración y otra puede presentar problemas muy importantes en la simulación, según el autor de esta librería. Un problema conocido, se presenta cuando en dos iteraciones consecutivas la diferencia entre el tamaño del paso de tiempo de ambas iteraciones es muy grande. Si en la primera iteración el paso de iteración es pequeño, y existe cualquier fuerza aplicada a un cuerpo, en la siguiente iteración esta fuerza será extrapolada en una fuerza de módulo mucho mayor que la del paso anterior, provocando problemas en la interacción de cuerpos. Por otro lado, utilizando pasos de tiempo variables, la simulación es no determinista, por lo que esto puede presentar un problema si el determinismo es una característica deseable. Utilizando un paso de simulación de largo fijo, puede ocurrir que el tiempo de procesamiento para avanzar la simulación sea mayor que el paso dado en la misma, siendo poco fiel a la realidad. Además esto implica

que la simulación dependa de la computadora en la cual se está ejecutando. Se puede lograr que la simulación sea razonablemente fiel al tiempo real con pasos de simulación de pasos fijos, pero para esto se debe realizar un control muy estricto en lo que concierne a la cantidad de veces que se avanza el paso de simulación en un determinado período de tiempo, corroborando la correctitud de la simulación conociendo el tiempo real del sistema. Estas limitaciones deben ser consideradas en el contexto de este proyecto, ya que la poca flexibilidad en el control del paso del tiempo es una limitación que se debe tener en cuenta.

2.2.6. Entorno de programación y plataformas

ODE está programado en C, pero la interfaz para su uso está en C++. Por otra parte, se pueden encontrar wrappers de esta librería para utilizarlo con distintos lenguajes, por ejemplo Odejava brinda la posibilidad de utilizar la librería anexándola directamente a un proyecto Java. Es multiplataforma, con soporte para Linux, Windows y MacOS.

2.3. Newton Game Dynamics

Esta librería es de código cerrado, sin embargo es gratuita. Según su documentación oficial que puede encontrarse en [NGD07], esta librería es rápida, estable y muy sencilla de utilizar. De forma muy fácil se pueden modelar objetos complejos como ragdolls (muñecos con características humanas) o vehículos con ruedas, manipulando todo tipo de características de los mismos. Esta librería prioriza la exactitud física por sobre la velocidad en la simulación. Otra 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, ya que la simulación puede ser fácilmente repetible. 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 librerías 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 usuarios discuten dudas y problemas.

A continuación se describen algunos conceptos importantes relacionados a esta librería.

2.3.1. Cuerpos Rígidos

Representan los objetos principales de la simulación y tienen asociada una primitiva de colisión, que es la que determina su forma. Además tienen un conjunto de propiedades configurables asociado, las más significativas de dichas propiedades se detallan a continuación.

- Matriz de transformación: Almacena las coordenadas del cuerpo en cada uno de los tres ejes. Se puede utilizar para rotar y trasladar el cuerpo, realizando operaciones sobre esta matriz.
- Matriz de masa: Es una cuaterna en la cual está especificada la masa del cuerpo, y su momento de inercia en cada uno de los tres ejes. Estos parámetros deben ser ajustados cuidadosamente para que se ajusten a la realidad de la manera más fiel posible.
- Vector de velocidad lineal: Análoga a la descripta en la sección 2.2.1.
- Vector de velocidad angular: Análoga a la descripta en la sección 2.2.1.
- Vector de fuerza: Representa la fuerza lineal neta a la que está siendo sometido el cuerpo en cada uno de los tres ejes.
- Vector de fuerza angular: Representa la fuerza angular neta a la que está siendo sometido el cuerpo en cada uno de los tres ejes.

2.3.2. Empalmes

Se utilizan para conectar cuerpos rígidos y eliminar uno o más grados de libertad en el movimiento relativo entre ambos. Los tipos de empalmes presentes en esta librería son:

- Ball and socket: Análoga a la descripta en la sección 2.2.2.
- Hinge: Análoga a la descripta en la sección 2.2.2.
- Slider: Análoga a la descripta en la sección 2.2.2.
- Corkscrew: Es un empalme similar al anterior pero con un mayor grado de complejidad. Le permite a uno de los cuerpos rotar a medida que se mueve a lo largo del eje de movimiento del empalme, tal y como lo haría un tornillo. Se puede configurar cuanto rota un cuerpo en relación a su movimiento lineal a lo largo del eje del empalme.
- Universal: Análoga a la descripta en la sección 2.2.2.
- UpVector: Es un empalme que restringe la rotación de un cuerpo a un eje determinado.
- Definidas por el usuario: Se pueden crear empalmes con hasta seis grados de libertad, tres lineales y tres angulares. Se utiliza en situaciones especiales en las cuales ninguno de los empalmes existentes logran adaptarse a las necesidades.

2.3.3. Callbacks

En la librería Newton se puede observar un paradigma orientado a eventos. Luego de crear un cuerpo, es posible asociar cierta información a este y además existe la posibilidad de asociar una función que se ejecuta cuando cambia algún tipo de información relacionada con este objeto. Se puede asociar funciones de callback a una variada cantidad de eventos diferentes que afectan a los cuerpos. Esto aporta gran flexibilidad, ya que se puede manejar gran cantidad de objetos sin tenerlos almacenados en ningún tipo de estructura en la aplicación del usuario, solamente manteniendo en el objeto la información necesaria sobre sí mismo y las funciones de callback para realizar las acciones correspondientes en los eventos deseados.

2.3.4. Colisiones

En Newton no existe el concepto de espacios. Están soportadas siete tipo de primitivas simples de colisión, las mismas se detallan a continuación.

- Box: Análoga a la descripta en la sección 2.2.4
- Sphere: Análoga a la descripta en la sección 2.2.4
- Cylinder: Representa un cilindro, al cual se le especifica su radio y altura
- Cone: Representa un cono, al cual se le especifica su radio y altura
- Capsule: Representa una cápsula, a la cual se le especifica su radio y altura
- Convex hull: Se utiliza en los casos en que no se puede representar lo que se desea utilizando las primitivas más simples. Crea una estructura convexa a partir de una lista de vértices especificada en el momento de su creación.
- Ray: Análoga a la descripta en la sección 2.2.4

Existen además dos tipos especiales de primitivas de colisión denominadas geometrías de colisión: "CollisionTree" y "User Defined Collision". En ciertas situaciones es recomendable utilizar alguna de estas geometrías, ya que utilizar las primitivas simples directamente no siempre resulta práctico y eficiente. No resultan tan flexibles como las demás primitivas, ya que no se les puede aplicar fuerzas o velocidades directamente, y si se desea modificar su ubicación y orientación, esto debe llevar a cabo operando directamente con su matriz de transformación. Cuando estas geometrías son asignadas a un cuerpo rígido, se anula la masa de dicho cuerpo, transformándolo en un cuerpo estático. La gran ventaja que presentan es que pueden poseer cualquier forma que se desee, ya que puede estar formada por una cantidad casi ilimitada de polígonos.

2.3.5. Materiales

El manejo de las interacciones entre los cuerpos se realiza tomando en cuenta los materiales de los mismos. Se representa el conjunto de materiales como un grafo no dirigido, en el cual los nodos son los identificadores de los distintos materiales y las aristas almacenan la información respecto a la interacción entre dos materiales. Se deben crear todos los materiales existentes en la simulación y luego para cada par de materiales posible se debe especificar las propiedades de su interacción, modelando así el comportamiento de puntos de contacto entre ambos materiales. Se debe especificar la dureza, la elasticidad y la fricción en dichos puntos de contacto. Adicionalmente se debe especificar si ambos materiales deben colisionar o no, y especificar una función de callback que se ejecuta cuando esto ocurre. Aparte, existe un material por defecto, que se le asigna a los cuerpos que no están asociados a ningún material.

Este enfoque para manejar las propiedades de las colisiones tienen como ventaja la facilidad en su utilización y ajuste, sin embargo se debe conocer de antemano todos los posibles materiales que van a existir en la simulación.

2.3.6. Otras características

No se puede especificar directamente la gravedad en esta librería, sino que para modelarla se debe especificar una fuerza con las características correctas y aplicarla a todos los cuerpos. El paso de simulación es especificado obteniendo a través de una operación cuanto tiempo ha transcurrido desde la última actualización de la simulación y se avanza la misma utilizando este lapso de tiempo. En ocasiones, es deseable manipular este período de tiempo. Por ejemplo, avanzando la simulación un período el doble mayor al tiempo transcurrido entre cada paso de simulación, se lograría avanzar la simulación al doble del tiempo real, aunque esto implicaría una serie de otros inconvenientes, ya que se perdería precisión en los cálculos y la simulación se alejaría más de la realidad de lo que lo haría en caso de utilizarse un paso de simulación más pequeño.

2.3.7. Entorno de programación y plataformas

Está hecho enteramente en lenguaje C. Es multiplataforma, con soporte para Linux, Windows y MacOS.

2.4. Ageia PhysX

Es extensamente utilizada en el desarrollo de juegos. Es una librería cerrada y tradicionalmente se vendió para proyectos comerciales y se utilizaba de forma libre en proyectos no comerciales, pero recientemente ha cambiado esta política y se volvió gratis para cualquier propósito. 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. Según [AGBR] presenta una muy interesante combinación de velocidad, precisión, robustez y estabilidad, transformándola en uno de los

motores físicos más fieles a la realidad. Se pueden configurar muchas características de la simulación, como por ejemplo las unidades de masa y medida con las que se trabaja.

A continuación se describen algunos conceptos importantes relacionados con esta librería.

2.4.1. Cuerpos Rígidos

Presentan el siguiente conjunto principal de características:

- Masa: Valor de la masa del cuerpo
- Vector Posición: Análogo al descripto en la sección 2.2.1.
- Vector Velocidad Lineal: Análogo al descripto en la sección 2.2.1.
- Vector Fuerza Lineal: Análogo al descripto en la sección 2.3.1.
- Matriz de Inercia: Análogo al descripto en la sección 2.2.1.
- Vector Velocidad Angular: Análogo al descripto en la sección 2.2.1.
- Vector Fuerza Angular: Análogo al descripto en la sección 2.3.1.
- Matriz de orientación: Describe la orientación del cuerpo, es representada por una matriz 3x3.

2.4.2. Empalmes

Esta librería presenta una gran cantidad de empalmes con el objetivo de modelar la mayor cantidad posible de situaciones.

Los empalmes soportados son:

- Revolute: Análogo al empalme "Hinge" descripto en la sección 2.2.2.
- Spherical: Análogo al empalme "Ball and socket" descripto en la sección 2.2.2.
- Cylindrical: Análogo al empalme "Slider" descripto en la sección 2.2.2.
- Fixed: Análogo al descripto en la sección 2.2.2.
- Distance: Mantiene cierta distancia especificada entre dos cuerpos rígidos
- Point-In-Plane: Restringe a uno de los cuerpos asociados al empalme, a que se mueva a lo largo de un plano que contiene al otro cuerpo rígido asociado a dicho empalme. Permite a los cuerpos involucrados rotar en cualquier eje pero solo trasladarse a lo largo de dos ejes.

- Point-On-Line: Restringe a un cuerpo a que se mueva a lo largo de una línea fijada al otro cuerpo rígido con el cual está conectado. Se puede pensar como un empalme de tipo "Spherical", el cual se puede trasladar a lo largo de un eje.
- Pulley: Su funcionamiento es similar al de una cuerda.
- 6DOF: Este empalme presenta seis grados de libertad, el cual puede ser configurado estableciendo límites y restricciones en cada uno de los grados de libertad, restringiendo el movimiento de los cuerpos rígidos involucrados.

Es importante destacar que los empalmes se pueden romper, y cuando ocurre esto lo hacen de manera definitiva. Se puede especificar si un empalme es rompible o si no lo es, y en caso de que sea, se debe especificar la fuerza y el torque máximo que soporta antes de romperse. Otra característica importante de los empalmes en este motor físico es lo que se denomina proyección. Cuando un empalme presenta un error mayor a un valor dado, siendo este un error muy importado que no puede ser corregido mediante pequeñas fuerzas correctoras, las posiciones de los objetos que este empalme relaciona serán proyectadas hacia una posición en la cual cumplan las restricciones del empalme. Esto solamente está soportado por los empalmes 6DOF, Revolute y Spherical.

2.4.3. Colisiones

Ageia PhysX permite modelar muchos tipos de interacciones utilizando el concepto de materiales y además permite realizar gran cantidad de optimizaciones en grandes simulaciones, haciéndolo un poco complejo de manejar en estos casos pero permitiendo simulaciones muy realistas. Las primitivas de colisión que maneja son:

- Box: Análoga a la descripta en la sección 2.2.4
- Sphere: Análoga a la descripta en la sección 2.2.4
- Cylinder: Representa un cilindro, al cual se le especifica su radio y altura
- Plane: Análoga a la descripta en la sección 2.2.4
- Capsule: Análoga a la descripta en la sección 2.3.4
- Convex Mesh: Crea una estructura convexa compleja que no puede ser representada por las primitivas anteriores. Existen dos alternativas al momento de dicha estructura:

    1. Utilizando una nube de puntos especificando cada vértice de la estructura
    2. Especificando un conjunto de triángulos que cumplen ciertas características, como que no haya ninguno con área cero o que la estructura generada por dichos triángulos sea convexa.

- Heightfield: Se utiliza para crear el terreno de la simulación. Se puede determinar esta primitiva especificando triángulos como en la Convex Mesh pero esto menoscaba el rendimiento de la simulación, por lo que lo usual es crear esta primitiva especificando un conjunto de rectángulos.

Se pueden realizar combinaciones de estas primitivas, fusionándolas para crear un objeto nuevo, creando así todo tipo de formas.

Un concepto importante a tener en cuenta en el sistema de colisiones de este motor físico es el de concepto de grupos. Los distintos objetos se pueden asociar entre ellos en grupos. Este concepto es similar al concepto de "Espacios" en ODE. Se utiliza cuando se desea que objetos de un grupo no colisionen con objetos de otro grupo, de esta manera no se realiza la detección de colisiones entre estos dos grupos, ahorrando poder de cálculo y optimizando el rendimiento del sistema. Se puede especificar para cada par de grupos si se desea detectar y efectuar las colisiones entre ambos grupos o no.

Un problema importante de muchos motores físicos es que objetos lo suficientemente chicos que se mueven a gran velocidad pueden pasar a través de otros objetos, pues no se detecta la colisión. Esto se debe a que en un paso de simulación, el objeto se encontraba en una posición anterior a la colisión con el otro objeto y en el siguiente se encuentra luego de lo que tendría que haber sido la colisión pero ya se pasó de la posición de impacto con el cual debería colisionar. Para resolver esto, este motor físico presenta un sistema denominado "CCD[2]". Para que la simulación incorpore este sistema, el mismo debe estar activado y solamente los objetos que se desea van a utilizar este tipo de detección de colisiones.

Además, se puede especificar el ancho de la piel de los cuerpos en general. Esta medida indica cuanto dos cuerpos pueden llegar a penetrarse en una colisión. Es un parámetro muy importante y debe ser analizado con cuidado, ya que se puede regular de acuerdo a cuan reactivas se desea que sean las colisiones de la simulación.

2.4.4. Materiales

El concepto de materiales en esta librería es similar al de la librería "Newton Game Dynamics", aunque un poco más flexible y complejo. Resultan determinantes en las características de la colisión entre dos objetos. Modelan no solo el material de la superficie de un objeto, sino también el material con el cual están hechos los objetos en su interior. Una propiedad fundamental de los materiales es su coeficiente de restitución, cuyo valor influye en como interactúan los cuerpos. Cuanto mayor sea este coeficiente, menor será la cantidad de energía perdida en un impacto, o sea, rebotará con más energía. Otro concepto importante es el de fricción. Cuanto mayor sea la fricción de un material, mayor energía perderá el objeto de ese material al deslizarse sobre otro objeto, frenándose cuando esto ocurre. Se puede especificar tanto la fricción estática como la dinámica de un

[2]Continuous Collision Detection

material, o sea, la fricción cuando el objeto no se está moviendo con respecto al objeto con el cual colisiona, y la fricción cuando se está moviendo.

2.4.5. Otras características

Se debe especificar la fuerza de gravedad en la simulación. Por otra parte, en cada paso de simulación se debe indicar cuanto tiempo se desea avanzar la simulación, siendo generalmente este período el intervalo de tiempo transcurrido desde la última actualización. Este intervalo es variable, sin embargo el motor físico divide a este período en unidades de tiempo más pequeñas que puede ser fijada y al avanzar determinado intervalo de tiempo la simulación, se avanza sucesivas veces esta pequeña unidad de tiempo. De esta manera se puede obtener una simulación determinista y repetible al fijar dicha unidad de tiempo en un valor dado. Esto aporta flexibilidad al manejo del paso de tiempo de la simulación y puede ser muy útil en el contexto de este proyecto.

Además, es importante mencionar que presenta la funcionalidad de almacenar el estado de una simulación en un archivo, o cargar una simulación previamente guardada.

2.4.6. Entorno de programación y plataformas

La librería Ageia PhysX está programada en C++. No es multiplataforma, está soportada solamente en entorno Windows.

2.5. Havok

Este es un motor físico cerrado y comercial, no es de libre uso para ningún tipo de proyecto. Es utilizado por el simulador oficial de la categoría Simurosot de la FIRA, que se describirá en la sección 5.2.

La información sobre esta librería está únicamente disponible para desarrolladores licenciados, por lo que la información sobre la misma presente en este documento fue enteramente obtenida en el artículo [HH07].

2.5.1. Cuerpos Rígidos

Presentan las siguientes propiedades:

- Masa
- Elasticidad
- Fricción
- Coeficiente de restitución

2.5.2. Empalmes

Todos los parámetros de configuración de los empalmes pueden ser editados en tiempo de ejecución. Se pueden especificar límites, restricciones y fricción a los mismos. Los tipos de empalmes soportados son:

- Point to Point: Un punto de un objeto se relaciona con un punto de otro objeto.
- Point to Nail: Un punto de un objeto se relaciona con un punto en el espacio.
- Hinge: Análogo al descripto en la sección 2.2.2.
- Ragdoll: Tipo especial de empalme que simula las articulaciones de un humano.
- Car wheel: Tipo especial de empalme que facilita la tarea de modelar ruedas de vehículos.
- Stiff Spring: Similar al empalme "Point to Point" en su funcionamiento, pero más eficiente en términos computacionales.
- Spring: Como un resorte al cual se le controla el largo máximo y otras propiedades
- Dashpots: Similar al empalme "Spring" pero actúa modificando la velocidad de los objetos, no efectuando fuerzas.

2.5.3. Colisiones

Las primitivas de colisión soportadas son:

- Spheres: Análogo a la descripta en la sección 2.2.4.
- Planes: Análogo al descripto en la sección 2.2.4.
- Convex objects: Representa un cuerpo convexo con cualquier forma.
- Concave objects: Representa un cuerpo cóncavo con cualquier forma.
- Fixed Polygon Soups: Es utilizado para modelar objetos fijos que nunca cambian, por ejemplo puede ser utilizado para modelar partes del terreno como montañas.
- Heightfields: Análogo al descripto en la sección 2.4.3.
- Definidas por el usuario

Objetos que no causan colisiones son denominados objetos fantasmas, los mismos resultan buenos para modelar sensores.

2.6. nV Physics

Este motor físico [NVPHY] se ha gestado hace muy poco tiempo y aún se encuentra en versión beta. Su desarrollo comenzó a principios del 2004. Presenta documentación muy básica, es muy fácil de utilizar pero no tiene grandes prestaciones. Además, según [NVWIK] su rendimiento en cuanto al manejo de colisiones es aún muy pobre. Se dispone de muy poca información sobre esta librería, y su sitio web no ha estado disponible en las últimas semanas, por lo que es probable que el proyecto no esté activo.

Los cuerpos rígidos están determinados por su forma y masa. Se los puede manipular aplicando fuerzas y torques. Las primitivas de colisión soportadas son las siguientes:

- Spheres: Análogo a la descripta en la sección 2.2.4.
- Boxes: Análogo al descripto en la sección 2.2.4.
- Planes: Análogo al descripto en la sección 2.2.4.
- Convex Triangles Meshes: Análogo a la primitiva "Convex Hull" descripta en la sección 2.3.4.
- Concave Triangle Meshes: Análogo a la primitiva "Concave object" descripta en la sección 2.5.3.

A las primitivas de colisión se les puede especificar características como fricción y elasticidad.

Los empalmes soportados son los siguientes:

- Ball and socket: Análogo al descripto en la sección 2.2.2.
- Hinge: Análogo al descripto en la sección 2.2.2.
- Hinge-2: Análogo al descripto en la sección 2.2.2.
- Universal: Análogo al descripto en la sección 2.2.2.
- Static: Análogo al empalme "Fixed" descripto en la sección 2.2.2.
- Car wheel: Modela una rueda de un vehículo, es similar a utilizar el empalme "Hinge-2".

Es muy escasa o inexistente la documentación sobre las características de cada uno de estos empalmes en este motor físico.

Se debe especificar el paso de simulación cada vez que se avanza la misma.

2.7. Otros

Existen diversos motores físicos disponibles. Se puede nombrar por ejemplo Bullet Physics Library[BULL], Open Tissue[TAXIS], Tokamak[TOKA], True Axis[TAXIS]. Los mismos comparten cualidades y defectos con algunos de los motores descriptos anteriormente.

Bullet es gratis y de código abierto. Utiliza la librería ODE en algunas de sus funciones. Presenta documentación no muy extensa ni exhaustiva y pocos ejemplos. Presenta un diseño modular en C++ y es relativamente sencillo de utilizar.

Tokamak también es gratis pero código cerrado. Se enfoca principalmente en la creación de juegos. Presenta escasa documentación sin embargo tiene diversos ejemplos y se pueden encontrar varios más en la red. Tiene una interfaz orientada a objetos en C++. Una de las principales desventajas es que solamente se lo puede utilizar en Windows.

Open Tissue es una colección de trabajos de diferentes personas integrados en un motor físico. Una gran ventaja que presenta es el hecho de ser una librería muy extensa, abarca gran cantidad de tipos de objetos diferentes, empalmes y otras características, incluyendo cuerpos deformables. Está escrita en C++ siguiendo un paradigma de orientación a objetos, sin embargo el estilo que sigue lo hace difícil de utilizar en un principio. Presenta muy escasa documentación, lo cual la pone en desventaja respecto a otras librerías. Es multiplataforma, siendo soportada por Windows y Linux.

True Axis es un motor físico gratis para proyectos no comerciales. Está programado en C++. Está diseñado priorizando la velocidad y de esta manera sacrifica el realismo. La documentación es relativamente buena, con diversos ejemplos de uso. Puede ser utilizado en Windows y Linux.

3. Motores Gráficos

3.1. Introducción

Una característica deseable en este proyecto es lograr independizar el componente físico y el componente gráfico del simulador. Esto lograría una mayor flexibilidad, ya que en caso de sustitución del motor físico, esto no afectaría al componente gráfico de la aplicación y viceversa. Además se podría implementar de tal manera que hayan varios motores gráficos, cada uno independiente del otro y trabajando sobre el mismo modelo físico. Para esto se considera importante evaluar algunos de los motores gráficos gratuitos disponibles en la actualidad. Se busca características como una buena documentación, muchos ejemplos de su utilización, portabilidad, facilidad de uso, entre otros.

3.2. OGRE 3D

OGRE[3][OGRE] es un motor gráfico de código abierto y de alto rendimiento escrito en C++, siguiendo el paradigma de orientación a objetos. Es totalmente gratis, fácil de utilizar y muy bien documentado. Se puede configurar para que utilice tanto Directx como OpenGl. Es ampliamente utilizado tanto en juegos como en simulaciones, siendo el motor gráfico de código abiero más utilizado, por lo que existe gran cantidad de información y ejemplos en la red. Los conceptos fundamentales que constituyen los pilares de esta librería son:

- Manejador de Escenas: Se encarga de manejar todos los objetos de importancia en el modelo gráfico. Tiene referencias a todas las entidades, cámaras, luces, entre otras cosas. Existen varios tipos de manejadores de escenas y cada uno tiene un propósito específico.
- Entidades: Es un tipo de objeto que puede se dibujado en una escena. Se puede pensar como un objeto que tiene una representación tridimensional en la pantalla.
- Nodo de Escena: Una característica importante de Ogre es que no es posible situar entidades directamente en escenas, sino que se debe relacionar una entidad con un nodo específico de la escena. Estos últimos tienen la información acerca de las coordenadas en las que se encuentra. Una vez que se relacionan estos dos objetos, la entidad podrá ser desplegada en la pantalla. Se pueden relacionar nodos de escena con otros nodos, y formas jerarquías con ellos. De hecho en cada manejador de escenas existe un nodo raíz con el cual todos los demás nodos están relacionados. Estos nodos almacenan información acerca de la orientación de la entidad relacionada, por lo que se puede rotar, redimensionar y efectuar otras transformaciones para alcanzar el estado deseado.

[3] Object-Oriented Graphics Rendering Engine

3.2.1. Cámaras, sombras y luces

Las cámaras son el instrumento utilizado para visualizar las escenas creadas. El uso básico de una cámara es tan simple como crearla, especificar su ubicación y indicar un punto en el espacio hacia el cual se debe apuntar.

Existen tres tipos diferentes de sombras. Cuando se desea crear una sombra, se debe especificar el tipo de la misma. La diferencia entre estos es la complejidad y el costo en términos de hardware de cada uno y deben ser correctamente elegidos de acuerdo a la situación.

Las luces se crean muy fácilmente y se debe especificar el tipo de luz y una serie de parámetros de ajuste para lograr el funcionamiento deseado. Los tipos de luces son:

- Puntual: Un punto en determinada ubicación emite luz en todas las direcciones.
- Proyector: Luz proyectada a la cual se indica una posición, una dirección y un ángulo de apertura y se emite luz con esas características.
- Direccional: Una fuente de luz a distancia infinita emite luz con determinada dirección.

Puede existir un número ilimitado de luces en una escena.

3.2.2. Otras características

Se puede muy fácilmente llevar a cabo tareas que en un principio pueden parecer complejas como crear el terreno, el cielo y agregarle niebla al entorno. Los materiales de los objetos se especifican utilizando un lenguaje script cuya sintaxis se puede ver como un pseudo-C++. Es multiplataforma, con soporte para Linux, Windows y MacOS.

3.3. Irrlicht

Irrlicht[IRRL] es un poderoso motor 3D gratuito, multiplataforma y de código abierto escrito en C++ siguiendo el paradigma de orientación a objetos. Sus características principales son la facilidad de uso, el hecho de ser rápido, extensible y estable. Soporta OpenGl, Directx y en caso de que ninguno de estas librerías esté disponible, utiliza su propio software para dibujar escenas, por lo que siempre existirá compatibilidad. Se pueden encontrar wrappers de esta librería para múltiples lenguajes, como C#, Java, Perl, Ruby, Basic, Python y Lua.

La visualización en Irrlicht se logra utilizando un grado jerárquico de escenas. Un concepto importante es el concepto de nodo de una escena. La escena es representada por un grafo, y los nodos asociados a dicho grafo se moverán de forma coordinada, ya que si un nodo se mueve, todos los que están conectados a el se moverán de la misma manera. Un nodo puede ser un objeto, una cámara, luces, entre otras cosas.

Presenta buena documentación y variedad de ejemplos. Además hay manuales de integración de esta librería con diversos motores físicos, incluyendo ODE, Tokamak y Ageia PhysX.

Es multiplataforma, con soporte para Linux, Windows y MacOS.

4. Otras herramientas de simulación

4.1. Phi

Phi Virtual Environment Core[DMPHI] es un framework para crear simuladores de robots. Es una herramienta código abierto que ha sido desarrollada desde 2003 por Daniel Monteiro Basso. La versión actual es la 3.0. Presenta diversas características que vale la pena destacar, las mismas de detallan a continuación.

- Modelo físico: Su modelo físico está basado en la librería ODE (descripto en la sección 2.2), por lo que hereda de la misma diversas cualidades.
- Scripting: Para la descripción de las escenas y de la simulación en general se utiliza un lenguaje script denominado Lua. Una característica importante del mismo es que soporta sockets, por lo cual es posible comunicar al mismo con otras aplicaciones desarrolladas en otros lenguajes.
- Los cuerpos rígidos soportados se corresponden a los cuerpos rígidos soportados por la librería ODE.
- Los empalmes soportados se corresponden a los empalmes soportados por la librería ODE.
- Colisiones: Las primitivas de colisión soportadas son básicamente las soportadas por la librería ODE, excepto "Capped Cylinder" y "Triangular Mesh".
- Materiales: Los materiales son objetos que determinan como se van a visualizar los cuerpos en la simulación. Los mismos presentan una gran variedad de parámetros para regular su apariencia.
- Arquitectura Cliente Servidor: El simulador se comporta como servidor y pueden haber diversos clientes, y en cada uno de los cuales una consola de visualización.
- Grabación de simulaciones: Se pueden fácilmente crear logs de simulaciones y luego reproducirlas.
- Visualización: Se pueden modificar diversos factores de la visualización. Se pueden crear cámaras y luces de diversos tipos. Se puede mover las cámaras para ver la escena desde diversos puntos de vista y distancias. Esto permite ver la escena desde cualquier lugar, y de esta manera detectar imprecisiones y errores en el modelo más fácilmente. La simulación puede ser visualizada en tiempo real a través de OpenGl o se pueden reproducir simulaciones previamente guardadas utilizando una técnica denominada Raytracer o trazado de rayos. Esta técnica establece objetos, puntos de luz y una cámara, y luego se intenta simular el comportamiento de la luz.

- Control del paso del tiempo: Es posible ejecutar la simulación en diversos modos. Se puede especificar un factor mediante el cual se ajusta el paso del tiempo. Si este factor es 1 la simulación se lleva a cabo en tiempo real, si es menor será más lento y si es superior a 1, se simulará más rápido. Además se puede establecer un modo de simulación en el cual la misma vaya tan rápido como el procesador lo permita, siendo esto útil para simular a la máxima velocidad posible en un determinado contexto. También se puede simular en el modo paso a paso, en el cual para cada paso de simulación se debe indicar al simulador que debe avanzar.

- Portabilidad: Phi se debe utilizar en Linux. Aunque se puede compilar y ejecutar en Windows, pero según la documentación disponible en [DMPHI],es poco probable que funcione correctamente.

- Documentación: La documentación es muy escasa. Existen muy pocos ejemplos de utilización de esta herramienta, principalmente en su versión más nueva. Sin embargo, existe una muy buena comunicación con el desarrollador de la misma y excelente disposición del mismo a colaborar.

5. Casos de estudio

5.1. GSim

GSim[4][ACGS03] es un simulador desarrollado en el marco del proyecto de grado del equipo FRUTo. La principal motivación para la construcción de este simulador fue contar con un entorno en el cual se lograra realizar tareas de aprendizaje automático. La motivación que llevó a su desarrollo fue el deseo de contar con una API que se pudiera utilizar para ejecutar simulaciones de forma programática, siendo así una alternativa al simulador proporcionado por la FIRA, en la cual no se puede llevar esto a cabo. Este simulador resuelve el movimiento de los robots y sus colisiones en forma geométrica, sin basarse en un motor físico externo que respalde la simulación. Sin embargo presenta numerosos puntos fuertes, como permitir establecer el estado inicial de los robots, permitir configurar diversos aspectos de la simulación incluyendo el paso de simulación y además permite ejecutar secuencialmente cada estrategia antes de avanzar el paso de simulación. A continuación se verán las principales características de GSim.

5.1.1. Arquitectura

El simulador está compuesto por los siguientes paquetes:

- gsim: es el componente principal del simulador. Realiza tareas de sincronización y se encarga de la correcta comunicación entre todos los demás módulos del sistema.
- engine: es responsable de realizar los cálculos necesarios para realizar la simulación.
- fira: su objetivo es proporcionar la misma interfaz que el simulador de la categoría Middle League Simurosot de FIRA.
- gui: a partir de la información recibida, muestra el estado de la simulación dibujándolo en la interfaz gráfica del usuario.
- team: componente que presenta algunas estrategias básicas ya implementadas.
- util: componentes comunes a todos los módulos.

En el simulador se ejecutan cinco hilos distintos:

- GSim: es el hilo de ejecución principal, el hilo padre de todos los demás. Se encarga de conectar y sincronizar el flujo de información entre todos los demás hilos
- Engine: es el hilo en el cual corre el motor de simulación

[4] Geometric Simulator

- LeftTeam: es el hilo en el cual se ejecuta la estrategia del equipo izquierdo
- RightTeam: es el hilo en el cual se ejecuta la estrategia del equipo derecho
- Gui: es el hilo en el cual se ejecuta todo lo referente a la representación en la pantalla del estado de la simulación.

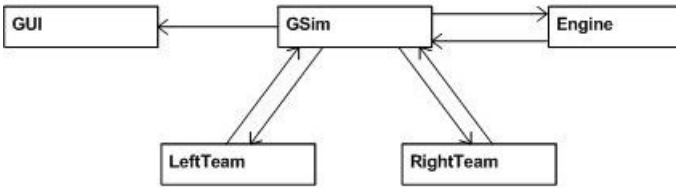

Figura 10: Hilos de ejecución de GSim y flujo de información entre los mismos (extraído de [ACGS03]).

La interfaz de comunicación del simulador con las estrategias está hecha de forma tal que el impacto de trasladar una estrategia desde el simulador GSim al simulador oficial de la Fira sea el mínimo posible. La comunicación del hilo GSim tanto con el hilo Engine como con los hilos de las estrategias es bidireccional y se lleva a cabo mediante el uso de colas de escritura bloqueante y lectura no bloqueante. La comunicación con la interfaz gráfica, sin embargo, es unidireccional y se utiliza la misma cola que en el caso anterior. Estas colas tienen capacidad para almacenar un elemento, por lo que puede ser sobrescrito en caso de no ser consumido. Otra característica importante es que el hilo Gui es tratado como un listener de la clase GSim, por lo que fácilmente puede ser reemplazada por otra interfaz gráfica o incluso se pueden agregar nuevos listeners y cada uno dibujar la interfaz gráfica de la manera que desee, ya que todos se ejecutan de manera concurrente. También el simulador puede funcionar sin que exista ningún listener que dibuje la interfaz gráfica. El hilo engine también es un listener de la clase GSim pero a diferencia de la interfaz gráfica la comunicación es bidireccional.

5.1.2. Modelo físico

Para modelar el comportamiento físico de la simulación se utilizó un enfoque híbrido, modelando las colisiones desde un enfoque geométrico y modelando el resto del comportamiento con un enfoque físico, o sea, basado en leyes físicas.

Se consideró al robot como una carrocería cúbica con dos ruedas independientes a ambos lados de la carrocería, controlada cada una por un motor. Se puede representar en cualquier momento el estado de un robot mediante su ubicación, orientación, velocidad lineal y velocidad angular. Tanto la velocidad lineal como la angular se pueden traducir en términos de la velocidad en cada una de las ruedas. El movimiento del robot, por lo tanto, está determinado por la velocidad relativa entre ambas ruedas. Además se estableció que debería existir una aceleración máxima y una desaceleración máxima para impedir que el robot cambie de velocidad instantáneamente, por lo que a cada ciclo de simulación se normalizan las velocidades de las ruedas para que las mismas cumplan con las restricciones de aceleración o desaceleración. Luego de normalizar a la velocidad, se debe ajustar los valores para que la velocidad relativa entre ambas ruedas sea realmente la deseada.

La pelota se modeló como una partícula, por lo que a diferencia de los robots no tiene rotación ni velocidad angular, de esta manera el estado de la pelota queda determinado por su posición y su velocidad lineal. Además para modelar el rozamiento de la pelota con la cancha, se utilizó una constante que afecta la velocidad de la misma.

Para modelar las colisiones se utilizó un modelo puramente geométrico. Se detectan colisiones de los robots con las paredes, de la pelota con las paredes y de los robots con la pelota, pero no de robots con robots. Para detectar colisiones de un robot con las paredes, se realiza una aproximación de la trayectoria del centro de masa del robot con un segmento de recta, de manera tal que los extremos del segmento coincidan con las posiciones del centro de masa de robot antes y después de avanzar el paso de simulación. Además se modela cada pared con un segmento de recta cuyos extremos coinciden con las ubicaciones en las cuales se intersecan con otra pared. Intersecando los segmentos de recta que representan paredes con los segmentos de recta que representan la trayectoria de los robots, se detectan puntos de colisión. Una vez detectado el punto de colisión se verifica el ángulo que forman el robot con la pared que va a colisionar. Sea alfa este ángulo, si alfa pertenece al intervalo [0, 45], el robot va a quedar con una orientación paralela a la pared. Si alfa pertenece al intervalo (45, 90] el robot va a quedar con una orientación perpendicular a la pared. En ambos casos se ajusta la posición del robot para que este quede dentro de los límites de la cancha y pegado a la pared.

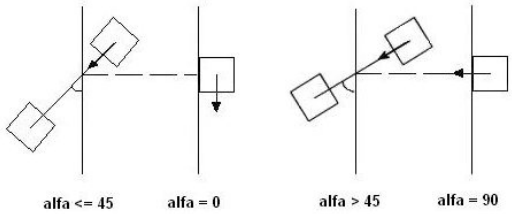

Figura 11: Colisión entre un robot y una pared. La orientación del mismo luego del choque depende del ángulo (alfa) con el cual éste colisiona con la pared (extraído de [ACGS03]).

Para resolver las colisiones entre las paredes y la pelota se utiliza un enfoque similar. En primer lugar se interseca los segmentos que representan las paredes y un segmento que representa el movimiento de la pelota para detectar puntos de colisión. Se considera que la colisión es completamente elástica, o sea, no se pierde energía. Lo único que se hace cuando ocurre el impacto es cambiar la dirección de alguna de las componentes de la velocidad de la pelota, y ajustar la posición de la pelota para que se encuentre dentro de los límites de la cancha.

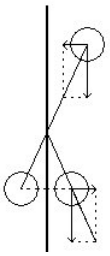

Figura 12: Colisión entre la pelota y una pared (extraído de [ACGS03]).

Para resolver las colisiones entre los robots y la pelota se utiliza una técnica similar a la utilizada para resolver las colisiones entre la pelota y las paredes, con la diferencia que en este caso se sustituye la pared por uno de los lados del robot. En primer lugar se calcula la distancia entre el centro de masa del robot y el centro de masa de la pelota. Si esta distancia es menor a un valor determinado por la mitad de la diagonal del robot más el radio de la pelota entonces el robot

y la pelota pueden estar colisionando. Luego se intenta determinar qué lado del robot está potencialmente chocando contra la pelota. Para determinar esto se calcula la distancia de cada vértice con el centro de masa de la pelota y se considera que el lado del robot que está potencialmente chocando la pelota es aquel cuyos dos vértices tienen menor distancia a la pelota. Una vez determinado el lado del robot, se realizan cálculo para verificar si el robot y la pelota están atravesándose, y en caso que así sea, se modifica el estado de la pelota de manera análoga a que si hubiese rebotado con la pared. El único caso singular ocurre si la pelota se aproxima al robot con una trayectoria que se corresponda exactamente a una de las diagonales del robot. En este caso las dos componentes de la velocidad de la pelota cambian de signo, saliendo en este caso la pelota hacia atrás por la misma trayectoria por la cual se aproximó al robot.

5.2. 3D Robot Soccer Simulator

3D Robot Soccer Simulator[RSOCF] es el simulador oficial de la categoría Simurosot de la FIRA[FIRA]. Su código es cerrado por lo que se tiene muy poca información acerca del funcionamiento interno del mismo. Está implementado utilizando el motor físico Havok, Shockwave 3D y lenguaje C. Presenta una interfaz gráfica en la cual se puede observar el campo de juego con los robots y la pelota. Además hay una variedad de opciones para controlar los partidos. Se puede ver el marcador actual y el tiempo de juego, que pueden ser modificados. Se puede pausar en cualquier momento la simulación y tomar alguna acción en el partido, como cobrar un penal, tiros libres, etc. Además existen repeticiones, en las cuales se puede ver lo que sucedió en el partido desde distintos ángulos y con diversas funcionalidades para controlar el paso del tiempo en la repetición.

Figura 13: Interfaz gráfica de Robot Soccer Simulator.

Una de las limitaciones de este simulador es el hecho de que las estrategias solo puedan estar implementadas en lenguaje Lingo o en C++. Para cargar las estrategias, se debe ingresar para cada equipo el nombre del archivo con la implementación de cada estrategia. En caso de estrategias implementadas en Lingo, estas deben estar en un archivo de texto y se ingresa el nombre de este archivo. En caso de estrategias implementadas en C++ las mismas deben estar compiladas en una DLL. Las estrategias son llamadas 60 veces por segundo por el simulador.

5.3. Robocup Soccer Simulator [SROBC]

Este simulador es utilizado en la liga simulada 2D de la RoboCup[ROBCU]. Su arquitectura fue pensada basándose fuertemente en un paradigma multi-agente.

Existen dos tipos de agentes, los jugadores y los entrenadores. Los jugadores básicamente ejecutan dos tipos de acciones: primarias y concurrentes. Las acciones primarias solo pueden realizarse una vez en cada ciclo de simulación y la concurrentes pueden realizarse al mismo tiempo varias de ellas concurrentemente con una acción primaria. El simulador solo ejecuta una acción primaria aunque el jugador solicite la ejecución de más de una de ellas.

Los jugadores reciben información de su entorno mediante tres tipos de sensores, los mismos son: el sensor físico, el sensor visual y el sensor auditivo. El sensor físico brinda información como la energía actual del robot o su velocidad actual. El sensor auditivo provee al robot información referente a los mensajes enviados por otros jugadores y por el árbitro, estos mensajes están limitados tanto en calidad como en cantidad según parámetros configurables en el simulador. El sensor visual brinda información acerca de objetos cercanos al robot. Si el objeto se encuentra en determinado cono de visión, se puede obtener información acerca de la distancia y la dirección del mismo, pero si se encuentra más lejos pero dentro de otro radio de proximidad un poco mayor, se puede detectar al objeto pero sin información sobre distancias o direcciones.

Los jugadores presentan además características individuales, como la velocidad y energía. Esto da lugar a que los equipos puedan ser heterogéneos, con distintos tipos de jugadores.

Los entrenadores son usuarios privilegiados del simulador ya que ejercen cierto nivel de control sobre el simulador. Existen entrenadores online y offline. Los entrenadores online tienen la finalidad de obtener información de diversos aspectos del juego y así interactuar con los jugadores para mejorar así el juego del equipo. Los entrenadores offline no pueden ser utilizados en competencias, solamente para entrenar y preparar al equipo. Se utiliza para entrenar determinadas situaciones del partido por lo que presenta operaciones especiales para esto.

No en todas las iteraciones los jugadores pueden contar con información actualizada del mundo, ya que el ciclo de simulación y la información recibida por los agentes son actividades asincrónicas que se ejecutan a intervalos diferentes y configurables.

Existe un árbitro automático pero este no detecta todas las situaciones deseables, solamente situaciones triviales como goles. En caso de querer efectuar acciones en situaciones más complejas como atascamientos, debe haber un árbitro externo para contemplar estos casos. El árbitro puede cambiar el estado del juego en cualquier momento, informando de esto a los jugadores.

Una característica importante del simulador es la funcionalidad de grabar en un archivo toda la información referente a un partido durante la simulación. Luego es posible utilizando un reproductor de logs provisto por el simulador, reproducir partidos enteros, con diversas funcionalidades, como el avance o retroceso rápido de la simulación.

5.3.1. Arquitectura

El simulador tiene 3 módulos: Field Simulator, Referee y Message Board. El primero es el que se encarga de actualizar y mover los objetos en la cancha, así como también detectar y manejar las colisiones entre las mismas. Para realizar el modelado físico de la simulación, este módulo utiliza un enfoque geométrico, sin contar con un motor físico. El módulo referee oficia de árbitro del juego, controlando el modo de juego. Cuando detecta algún evento en el cual debe tomar acciones, lo hace comunicando el modo de juego al que debe pasar el

simulador. El módulo Message Board se encarga de la comunicación con los agentes del sistema.

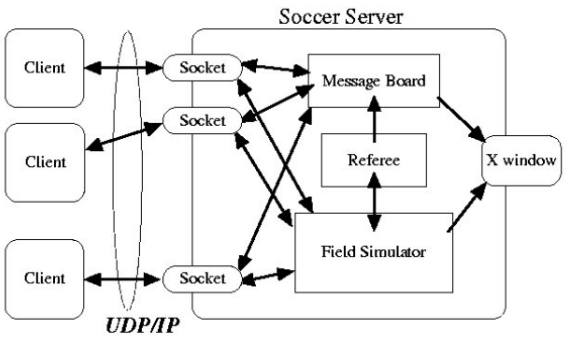

Figura 14: Arquitectura del simulador (extraído de [CCS05]).

El método de visualización de los partidos es independiente del funcionamiento del simulador. El mismo se comunica a través de sockets con agentes, que en caso de estar destinadas a la visualización se denominan Monitores. En particular en caso de que sea necesario se pueden agregar diversos monitores con el objetivo de visualizar el partido, o en el caso que no sea necesario visualizar el partido, el simulador puede funcionar perfectamente sin un monitor. En el monitor provisto por el simulador se puede ver la disposición de los objetos en el campo de juego, el marcador, el nombre de los equipos, entre otros datos.

5.4. MiS20 [MIS20]

El simulador MiS20[5] surgió como un intento por poseer una alternativa al simulador de la categoría MiroSot Middle League de Fira para perfeccionar la estrategia utilizada por el equipo Mi20[6]. Esta necesidad surge a partir de las carencias del simulador oficial como la necesidad de constante intervención humana en los partidos y una simulación con poca precisión. MiS20 presenta numerosas ventajas deseables en cualquier simulador. Se permite establecer las posiciones iniciales de los jugadores, calibrar diversos parámetros de la simulación, se puede controlar el paso de simulación, se puede entrenar diversas situaciones de juego y permite grabar y reproducir partidos. En analogía a GSim, tampoco posee un motor físico externo que lo respalde y resuelve colisiones de una manera similar.

[5] Mission Impossible Simulator
[6] Mission Impossible Twenty

5.4.1. Arquitectura[DF03]

Se utilizó una arquitectura basada en el modelo MVC[7], con el agregado de un componente de comunicación, por lo que los cuatro componentes serían modelo, vista, controlador y comunicación. A continuación se describirá cada uno de estos componentes.

El modelo es la parte fundamental del simulador, controlando todo lo referente a la coordinación, organización de los partidos y de la simulación en sí. Es también responsabilidad de este módulo manejar la grabación en archivos de estados de la simulación y una eventual reproducción a partir de estos archivos. Se utilizan archivos con formato XML[8], en los cuales se almacenan snapshots del estado de la simulación, logrando a partir de muchos de estos snapshots, escribir logs de partidos enteros que luego son reproducibles. Esta componente también posee la responsabilidad de manejar las detección y resolución de colisiones. Para detectar colisiones se realizan básicamente dos etapas. En la primera etapa se detectan parejas de objetos que potencialmente pueden estar colisionando. Esto se lleva a cabo con cálculos muy sencillos, utilizando circunferencias para representar objetos como los robots y la pelota y interceptando dichas circunferencias. A las parejas de objetos que en la primera etapa se infiere que puedan estar colisionando, se aplica la segunda etapa, con cálculos un poco más complejos para determinar si de hecho están colisionando. En la segunda etapa se representan las paredes y los robots con segmentos de rectas, estos últimos se representan utilizando cuatro segmentos, uno para cada lado. Las colisiones entre robots y paredes y entre robots y robots se detectan calculando la intersección entre estos segmentos. Las colisiones de la pelota con los demás objetos se detectan midiendo la distancia del centro de la pelota a los segmentos que representan los objetos, tomando en cuenta el radio de la pelota.

La interfaz gráfica fue desarrollada utilizando Java Swing, Java AWT y Java3D. Presenta un muy buen soporte para la interacción con el usuario así como una amplia cantidad de información acerca de diversos aspectos del partido y de la simulación. En todo momento se despliega información actualizada acerca del tiempo de juego, marcador del partido, la posición y velocidades de las ruedas de cada robots, así como también información acerca de eventos en el sistema. Se puede pausar o reanudar un partido previamente pausado, o incluso determinar acciones asumiendo el rol de juez, como cobrar tiros libros, penales o goles.

[7] Model-View-Controller
[8] Extensible Markup Language

Figura 15: Interfaz gráfica del simulador MIS20.

El componente de control es el encargado de notificarle al simulador de cualquier evento generado por el usuario, como presionar algún botón en el menú. Este componente se encarga de detectar dichos eventos e informar al simulador que ha ocurrido algo relevante, indicándole que acción ha detectado.

El componente de comunicación fue realizado para comunicar al simulador con la estrategia del equipo Mi20. Para esto se utilizó JNI[9] para comunicarse con la estrategia implementada en lenguaje C.

5.5. Simulator Bob

Simulator Bob es un entorno de simulación 3D diseñado para simular una amplia variedad de robots móviles. Utiliza ODE[Smi07] para realizar el modelado físico y OpenGL[OPENGL] para la visualización gráfica. Su interfaz gráfica fue realizada utilizando MFC[10], y por lo tanto se puede utilizar exclusivamente en sistemas Win32. Es una aplicación gratuita y de código abierto.

5.5.1. Componentes

El funcionamiento de este entorno de simulación se basa en la interacción de componentes, los mismos se pueden clasificar en:

- Mecánicos: son cuerpos rígidos conectados por empalmes.

[9] Java Native Interface
[10] Microsoft Foundation Classes

- Actuadores: permite aplicar fuerzas y torques a los cuerpos.
- Sensores: realiza mediciones siguiendo ciertos criterios.
- Controladores: módulos de software que pueden acceder a datos de sensores, actuadores y otros controladores.

Existen diversos tipos de componentes, cada uno con características específicas. Cada componente concreto es una instancia de un tipo más básico de componentes que agrupa ciertas funcionalidades proponiendo una interfaz. Los tipos básicos de componentes se detallan a continuación:

- SimMouseHandler: los componentes de este tipo actúan como manejadores de eventos del ratón sobre la interfaz gráfica.
- CameraComponent: los componentes de este tipo se corresponden a una cámara que visualiza la simulación.
- BodyComponent: los componentes de este tipo están compuestos por una representación visual (tipo ModelComponent) y una representación física (tipo GeomComponent) de un cuerpo.
- GlobalSimComponent: es un componente genérico que puede ser utilizado con cualquier propósito.
- ModelComponent: los componentes de este tipo modelan la representación visual de un cuerpo, estableciendo las características de la misma.
- SkinComponent: los componentes de este tipo describen la representación visual de la superficie de un cuerpo.
- GeomComponent: los componentes de este tipo describen la representación física de un cuerpo, modelando su forma y características.
- JointComponent: los componentes de este tipo modelan un empalme entre dos cuerpos.
- MaterialComponent: los componentes de este tipo describen las propiedades físicas de un material que puede estar asociado a cuerpos.
- PoseComponent, MotionComponent, InertiaComponent: los componentes de estos tipos proveen información acerca de la posición, movimiento y propiedades de inercia respectivamente acerca de un cuerpo al cual están asociados.
- ControllerComponent, SimControllerComponent: Modela un módulo de software que puede eventualmente estar ejecutándose en un hilo independiente y puede comunicarse con otros controladores. Ambos pueden realizar modificaciones sobre los cuerpos en el transcurso de la simulación pero su diferencia radica en que el segundo puede acceder a datos de la simulación mientras que el primero no.

5.5.2. Funcionalidades

Para controlar el paso de tiempo de la simulación solamente se presenta tres opciones: empezar, pausar o detener. En la primera opción simplemente se continúa la simulación desde el estado inicial o el último estado antes de una detención. Tanto la segunda como la tercera opción se pueden utilizar para interrumpir momentáneamente la simulación, pero su diferencia radica en que en la tercera opción todas las velocidades de los cuerpos se llevan a cero.

Se dispone de una funcionalidad de grabación de simulaciones, por la cual es posible grabar en un fichero AVI un vídeo de la simulación desde el punto de vista de la cámara activa en el momento.

Además se puede especificar el nivel de detalle y distintas opciones gráficas, de modo de ajustar la performance a las necesidades. Se pueden especificar distintos modos de representación gráfica, configuración de la iluminación, entre otras propiedades.

La simulación propiamente dicha se carga desde un archivo con formato XML. Este archivo se puede dividir en tres secciones, las mismas son:

- Lista de cámaras: se describen los distintos componentes de tipo cámara en la simulación. Si ninguno es especificado, se creará uno automáticamente. Estos componentes no pueden ser creados desde la interfaz gráfica.
- Configuración de la escena: se especifican una serie de parámetros importantes en la simulación. Si los mismos no son especificados, tomarán valores por defecto. Además es importante destacar que todos estos valores pueden ser modificados desde la interfaz gráfica en tiempo de ejecución.
- Lista de componentes: en esta sección se describe a la simulación en sí, especificando todos componentes que forman la misma. Se puede especificar cualquier cantidad de componentes de los tipos nombrados en la sección 5.5.1.

5.5.3. Simulación

El motor físico utilizado requiere que para cada punto de contacto detectado, se le deba especificar un coeficiente de fricción, un coeficiente de elasticidad, y la velocidad mínima requerida para que ambos cuerpos reboten al chocar entre sí. Estos parámetros deben ser especificados para todas las combinaciones posibles de materiales que eventualmente puedan chocar entre sí, dando lugar a una matriz de datos que podría crecer mucho si se utilizan muchos materiales. Por esta razón, se especifica cada valor una vez para cada material y luego en cada punto de contacto se asigna para los coeficientes de fricción y elasticidad los valores de los máximos entre los materiales que colisionan y para la velocidad mínima de rebote se utiliza el mínimo de los valores correspondientes a ambos materiales. No existen justificaciones físicas para esta simplificación, pero según [PSBOB] utilizando este método se obtiene un comportamiento aparentemente razonable.

Cuando un módulo necesita acceder a datos de la simulación, le envía un pedido al simulador indicándole el momento del cual este requiere información, este momento debe ser un instante de tiempo posterior al actual tiempo de simulación. Estos pedidos son almacenados en una lista ordenada por el valor de dicho momento del cual requieren información. Cuando el pedido con un tiempo más temprano difiere muy poco del tiempo actual de simulación (menos que cierto intervalo menor al mínimo paso de simulación), el acceso a los datos de la simulación es otorgado a este componente que pidió acceso y la simulación solo continúa una vez que dicho componente haya liberado el acceso. Si esto no ocurre en un intervalo de tiempo preestablecido, la simulación continúa y dicho componente pierde acceso a los datos de la misma. Si el pedido con un tiempo más temprano es mucho mayor al tiempo actual (mayor que cierto intervalo mayor al máximo paso de simulación), se realiza otra iteración en la simulación sin liberar el acceso a los datos de la simulación al componente que realizó el pedido. Al cabo de esta iteración se verifica nuevamente si corresponde otorgarle el acceso a los datos al componente que requiere información.

5.5.4. Extensibilidad

Para cada uno de los tipos de componentes descriptos en la sección 5.5.1, existen diversos componentes concretos que los implementan, los cuales son referenciados desde el archivo donde se carga la simulación. El concepto de extensibilidad de esta herramienta está asociado a la creación de nuevos tipos de componentes concretos que implementen los tipos básicos. Existe una herramienta denominada "Class Creator Bob" que proporciona una manera simple de crear plantillas que faciliten la creación de nuevos componentes concretos y acoplarlos a la aplicación. De esta manera es posible crear nuevos cuerpos con diferentes formas y propiedades, nuevos actuadores, sensores, entre otros.

5.5.5. Interfaz de usuario

A continuación se puede ver la interfaz gráfica de esta aplicación y una simulación de ejemplo en curso.

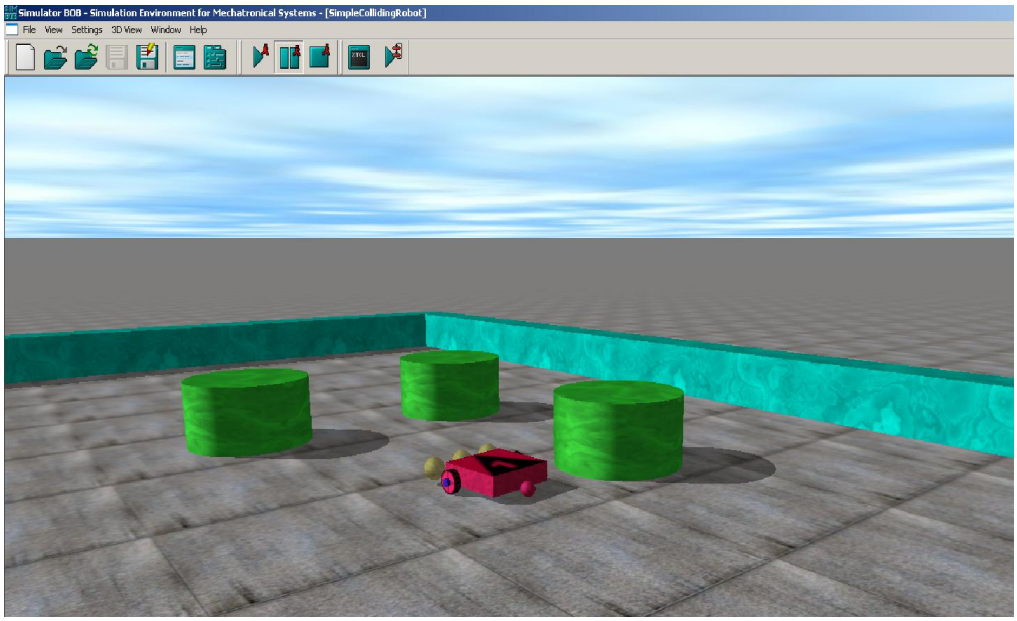

Figura 16: Interfaz gráfica del simulador Simulator Bob.

5.6. Webots

Webots es un avanzado simulador de robótica. Trae herramientas para definir modelos propios, definir las características de la simulación física y escribir controladores para los robots. Tiene versiones para Windows, MacOS y Linux, pero tiene la desventaja de ser una aplicación de código cerrado y no gratuita. Utiliza ODE[Smi07] para realizar el modelado físico y OpenGL[OPENGL] para la visualización gráfica.

5.6.1. Funcionalidades

Se pueden modelar robots y simulaciones tan complejas como soporte la librería física, ODE en este caso. El tiempo de simulación es muy flexible y puede ser ajustado de acuerdo a las necesidades. Es posible avanzar a la simulación paso a paso, a velocidad normal o a velocidad avanzada, en este último caso sacrificando precisión.

La información acerca del modelo que se desea simular es obtenida desde un archivo. Este archivo contiene toda la información acerca de los objetos que componen la simulación, sus propiedades físicas, actuadores, sensores, controladores de robots, propiedades de visualización y varias cosas más. Este archivo puede ser creado o editado desde la interfaz de usuario, a través de una interfaz amigable y fácil de utilizar, creando de esta forma los objetos deseados. Existen

una gran cantidad de actuadores y sensores disponibles que se pueden utilizar para que los robots tengan el comportamiento y releven la información deseada.

Es posible grabar vídeos de las simulaciones en ficheros AVI o WMV, pero no es posible grabar la simulación propiamente dicha, para luego reproducirla utilizando esta misma aplicación.

El comportamiento de los robots es manejado por controladores. Webots incluye interfaces en C, C++ y Java, a través de las cuales se puede programar directamente un controlador de un robot. A cada robot que se desee controlar, es necesario indicarle a la aplicación un ejecutable que contenga el programa de control, ya sea de extensión .exe en el caso de C y C++, o .class en caso de haberlo desarrollado en java.

La interfaz de usuario posee la funcionalidad de permitir el control mediante el mouse de los objetos de la simulación de manera muy sencilla, por lo que resulta útil para realizar simulaciones interactivas.

Esta aplicación además provee la funcionalidad de transferencia de controladores de robots hacia diversos tipos de robots reales comerciales existentes en la actualidad.

5.6.2. Interfaz de usuario

A continuación se puede ver la interfaz gráfica de esta aplicación y una simulación de ejemplo en curso.

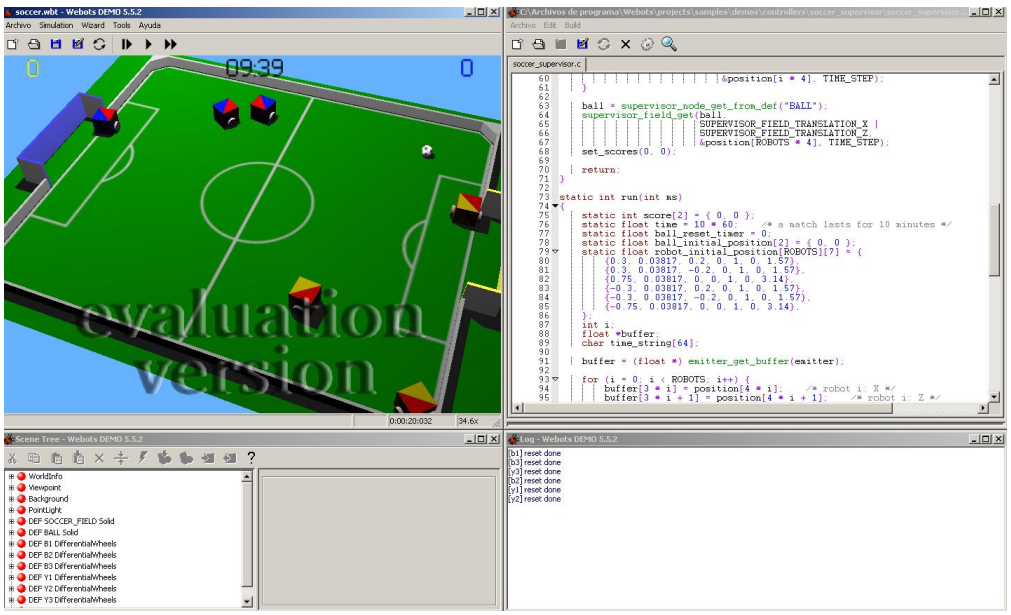

Figura 17: Interfaz gráfica del simulador Webots.

Referencias

[Smi07] Rusell Smith, Open dynamics engine v0.8 user guide, Mayo 2007.

[ABR05a] Gustavo Armagno, Facundo Benavides y Claudia Rostagnol, Robosoccer Engine, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, 2005.

[CCS05] Raúl Canales, Serrana Casella y Pablo Rodriguez, Fútbol de Robots: Liga de Simulación 2D Robocup, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay, Junio 2005.

[NGD07] Sitio Web Newton Dynamics Engine, http://www.newtongamedynamics.com, Mayo 2007.

[NWI07] Sitio Web Newton Wiki, http://walaber.com/newton_wiki/index.php, Mayo 2007.

[NHC07] Sitio Web http://www.runehunter.phpnet.us/NewtonHelpCode.html , Mayo 2007.

[AM06] Axel Seugling and Martin Rolin, Evaluation of Physics Engine and Implementation of a Physics Module in a 3d-Authoring Tool, Master Thesis, Department of Computing Science, Umea University, Sweden, Marzo 2006.

[HH07] Henrik Hansson, Craft Physics Interface, Master Thesis, University of Linkoping, Sweden, Enero 2007.

[ACGS03] Alvaro Castroman, Manual de usuario GSim, Setiembre 2003.

[FIRA] Sitio Web FIRA. http://www.fira.net, Junio 2007 .

[CHEN] Chen, Dorer, Foroughi, Heintz, Huang, Kapetanakis, Kostiadis, Kummeneje, Murray, Noda, Obst, Riley, Steffens, Wang, Yin. ,User Manual - RoboCup Soccer Server, versión 7.07 and laters - Febrero 2003.

[AGBR] Thiago Souto, Maior Cordeiro de Farias, Daliton da Silva, Guilherme de Sousa Moura, Márcio Augusto Silva Bueno, Veronica Teichrieb and Judith Kelner, Tutorial Ageia PhysX.

[AGDEV] Sitio web http://devsupport.ageia.com, Mayo 2007.

[AGEIA] Sitio web http://www.ageia.com, Mayo 2007.

[HAVOK] Sitio web http://www.havok.com, Mayo 2007.

[NVPHY] Sitio web http://www.thephysicsengine.com, Mayo 2007.

[NVWIK] Sitio web http://en.wikipedia.org/wiki/NV_Physics_SDK, Mayo 2007.

[BULL] Sitio web http://www.continuousphysics.com, Mayo 2007.

[OPTI] Sitio web http://www.opentissue.org, Mayo 2007.

[TOKA] Sitio web http://www.tokamakphysics.com, Mayo 2007.

[TAXIS] Sitio web http://trueaxis.com, Mayo 2007.

[DEVM] Sitio web http://devmaster.net, Mayo 2007.

[OGRE] Sitio web http://www.ogre3d.org, Mayo 2007.

[OGRW] Sitio web http://www.ogre3d.org/wiki/index.php/Main_Page, Mayo 2007.

[IRRL] Sitio web http://irrlicht.sourceforge.net, Mayo 2007.

[DMPHI] Sitio web http://www.lec.ufrgs.br/~dmbasso/phi/doku.php, Mayo 2007.

[ROBCU] Sitio web http://www.robocup.org, Mayo 2007.

[SROBC] Sitio web http://sserver.sourceforge.net/, Mayo 2007

[RSOCF] Robot Soccer v1.5a http://www.fira.net/soccer/simurosot/R_Soccer_v15a_030204.exe, Mayo 2007.

[MIS20] Sitio web http://hmi.ewi.utwente.nl/MiS20/, Mayo 2007.

[DF03] Hans Dollen y Wim Fikkert, MiS20 The Robotic Soccer Simulator, Junio 2003

[OPENGL] Sitio Web http://www.opengl.org, Octubre 2007

[PSBOB] Ing. Patrik Stellmann, Simulator Bob, 3D Simulation Environment for Mobile Robots, Febrero 2006

[SIMBOB] Sitio Web http://simbob.sourceforge.net/, Octubre 2007

[WBTS] Sitio Web http://www.cyberbotics.com/, Octubre 2007

[OMWB] Olivier Michel, Webots: Professional Mobile Robot Simulation.