Integración de Motores de Simulación y Motores de Visualización Aplicada a Simuladores de Entrenamiento

Horacio Abbate[1,2] and Agustín Mezzina[2]

[1] Instituto de Investigaciones Científicas y Técnicas para la Defensa CITEDEF (MINDEF)
habbate@citedef.gob.ar
[2] Universidad de Buenos Aires, Facultad de Ingeniería, Buenos Aires, Argentina
{habbate, amezzina}@fi.uba.ar

Resumen. En la industria de la simulación con fines de entrenamiento es de gran relevancia la generación de una representación visual del entorno simulado que presente un grado de realismo adecuado para garantizar la efectividad del ejercicio. No todas las herramientas disponibles en esta área presentan un sistema de visualización realista, o no son abiertas para el uso general. El objetivo de este trabajo es la integración de un sistema de modelado y simulación de sistemas físicos con un generador de imágenes tridimensionales de alta performance, utilizando herramientas abiertas y estandarizadas. En este documento, se analizan y se atacan los problemas que surgen de la comunicación de ambos sistemas en tiempo real y se diseña e implementa una solución basada en las técnicas estudiadas mediante la definición de un modelo físico de movimiento simulado en el entorno OpenModelica y su visualización a través de un generador de imágenes desarrollado con la biblioteca de gráficos tridimensionales OpenSceneGraph, comunicándose ambos mediante el protocolo CIGI (Common Image Generator Interface). El propósito de este trabajo es aplicar las herramientas desarrolladas inicialmente en proyectos de simulación de entrenamiento del Ministerio de Defensa, para finalmente abrirlas al resto de las áreas del Estado y a la comunidad.

Palabras clave: motores de simulación, motores de visualización, sistemas de entrenamiento, computación gráfica, software abierto.

1 Introducción

La inmersión de una persona en un entorno virtual a través de un sistema de simulación es de interés en varias áreas de la industria, entre ellas: aplicaciones militares y civiles, educación, industria, capacitación y entretenimiento. En estos entornos, se

presenta al usuario con una representación de un sistema, ya sea real o ficticio, con el que debe interactuar. Los estímulos que el usuario imparte al sistema son utilizados para realimentar el modelo físico y generar en él una respuesta adecuada.

Una parte clave de dicho sistema es la generación de gráficos tridimensionales que generen una representación visual realista del entorno que se está simulando, para maximizar el realismo de la experiencia. En ese sentido, la computación gráfica es un área de la informática que ha experimentado un avance significativo en los últimos años, tanto mediante avances tecnológicos aplicados a hardware dedicado, como aquellos relacionados a las técnicas de software utilizadas en la generación de imágenes. La variedad de estas técnicas ha logrado que el desarrollo de aplicaciones gráficas sean más accesible para el programador. Por otro lado, dentro del área de la simulación, existen diversas herramientas (lenguajes, frameworks, protocolos, etc.) desarrollados con el objetivo de modelar y simular situaciones de interés. Estas aplicaciones comenzaron utilizándose en la industria militar, y fueron extendiéndose a múltiples usos civiles. Las aplicaciones más comunes son el modelado de situaciones que permitan entrenar y desarrollar las capacidades tácticas y estratégicas, con diferentes niveles de realismo. No todas estas herramientas, sin embargo, presentan un sistema de visualización realista o son abiertas para el uso general.

El objetivo de este trabajo es integrar un sistema de modelado y simulación de sistemas físicos con un sistema de visualización de gráficos tridimensionales, utilizando herramientas open source y protocolos de comunicación abiertos y estandarizados. Para ello, se desarrollaron un modelo físico de comportamiento del sistema simulado utilizando el lenguaje Modelica en el entorno abierto OpenModelica y un generador de imágenes utilizando el framework abierto OpenSceneGraph. Ambos sistemas fueron provistos de la capacidad de intercambiar mensajes según el protocolo CIGI (Common Image Generator Interface), de forma que puedan funcionar de manera conjunta en tiempo real.

De este modo, se busca promover el armado de equipos multidisplinarios en el desarrollo de simuladores de entrenamiento, donde el personal encargado de desarrollar el modelo físico de las entidades estudiadas no requiera un conocimiento avanzado en la complejidad del sistema de visualización tridimensional, y viceversa. Por otro lado, la comunicación a través de un protocolo estandarizado permite la utilización del mismo modelo físico con distintos sistemas de visualización, o bien utilizar el mismo sistema de visualización para representar distintos modelos físicos.

2 Descripción del Sistema

2.1 Concepto

El esquema propuesto trata de un sistema distribuido donde se producen datos dinámicos en un host y se generan las imágenes (renderizado) en tiempo real en un host gráfico remoto. Estos datos se corresponden por ejemplo con el estado de movimiento (posición y orientación) de las entidades que participan del modelo físico simulado.

Dado que los sistemas de simulación de entrenamiento son altamente interactivos, el usuario espera una representación visual de las entidades remotas que presente un

conjunto particular de características, ellas son: precisión posicional, precisión de comportamiento y renderizado suave [1]. Precisión posicional significa que el error promedio de cada representación remota debe ser lo suficientemente pequeño para no afectar el grado de realismo requerido; precisión de comportamiento implica que la velocidad y aceleración de estas representaciones deben imitar el comportamiento de aquellas de la entidad original (del mondo real), aun cuando el valor específico no sea exacto. Por último, renderizado suave significa que la imagen debe animarse de manera fluida teniendo en cuenta la tasa de refresco local (un valor típico es de 60 cuadros por segundo), de manera de observar en todo momento una continuidad en el movimiento.

Cualquier imprecisión percibida por el usuario en este sentido genera confusión, acciones incorrectas y respuestas inconsistentes.

La transmisión de actualizaciones de estado a medida que son generadas según el ritmo del host emisor puede ser ineficaz debido a las limitaciones inherentes a la red, por ejemplo en cuanto a latencia y ancho de banda. No está garantizado que la actualización llegue ordenada ni a tiempo antes del renderizado del próximo cuadro, provocando una repetición de cuadros sucesivos. Por otro lado, si el ritmo de emisión de estados del host es menor que la tasa de refresco de la pantalla, el movimiento puede presentar saltos o perder fluidez.

Para resolver estos problemas, los hosts de visualización remotos generan y muestran en pantalla una predicción aproximada del estado de movimiento de las entidades, calculada a partir de información previa, que luego corrigen al recibir un estado de actualización.

Table 1. FÓRMULAS DE EXTRAPOLACIÓN

	Un paso	Dos pasos
1° Orden	$X_t = X_{t'} + v_t \Delta t$	$X_t = X_{t'} + \frac{X_{t'} - X_{t''}}{t' - t''} \Delta t$
2° Orden	$X_t = X_{t'} + v_t \Delta t + 0.5 a_{t'} \Delta t^2$	$X_t = X_{t'} + v_t \Delta t + 0.5 \frac{v_{t'} - v_{t''}}{t' - t''} \Delta t^2$

2.2 Dead reckoning

Dead reckoning es un método utilizado para limitar el ritmo con el que un host emite actualizaciones de estado basado en algoritmos de predicción. Fue definido dentro del protocolo de comunicación para sistemas de simulación distribuidos DIS (Distributed Information System) [2].

Al predecir el estado de movimiento de las entidades remotas, no es necesario recibir una notificación por cada actualización de estado. La misma se envía solamente cuando el estado real de una entidad difiere del predicho por un valor mayor a un umbral de tolerancia preestablecido apropiado para la aplicación de interés.

Para esto, cada host debe mantener tanto el estado real de sus entidades, como el predicho para ellas por el algoritmo, y debe corregir esta predicción cada vez que se

supere el valor umbral.

El host remoto debe determinar el estado de las entidades simuladas utilizando la posición y orientación que predice para ellas en cada cuadro. Al recibir una actualización, debe realizar la misma corrección que el host de origen y actualizar la información del modelo. Esta corrección se realiza utilizando técnicas de suavizado, de manera que el cambio no sea apreciable visualmente para el usuario. Por esta razón, el algoritmo posee dos etapas: una de predicción y una de corrección.

La predicción propuesta en el algoritmo se basa en asumir que la velocidad o aceleración de las entidades permanecen constantes entre cada actualización de estado, por lo que los nuevos estados son calculados mediante una función de extrapolación acorde (Tabla 1).

Se puede realizar un cálculo de primer orden tomando en cuenta una velocidad constante, o un cálculo de segundo orden tomando en cuenta una aceleración constante. A su vez, estas dos magnitudes pueden, o bien ser informadas por la entidad en sus actualizaciones, o bien ser estimadas por el algoritmo a partir de estados anteriores, utilizando una fórmula multi-paso.

Para corregir la posición del modelo en la etapa de convergencia, el algoritmo propone suavizar la transición hacia el nuevo estado aproximándose en forma lineal a través de n iteraciones, de manera que no se aprecien saltos en la visualización.

En cada iteración, la nueva posición es:

$$Pi = P0 + (Pf - P0)\ i/n \tag{1}$$

En la ecuación (1), i es un entero que va desde 1 hasta n, $P0$ es la posición antes de la actualización y Pf es la posición final, es decir, la recibida en la actualización.

2.3 Compensación de latencia

Al llevar a cabo un ejercicio de simulación distribuido como el descripto, un factor que puede afectar la percepción sensorial del usuario está relacionado a la velocidad de los enlaces de red. Un mensaje de actualización proveniente de una entidad remota debe viajar a través de la red para llegar a su destinatario, lo que introduce una latencia en la comunicación. Esto significa que cuando un host recibe la información, la misma ya está, en mayor o en menor medida, desactualizada. Realizar una corrección del modelo predicho sin tener en cuenta esta latencia puede introducir efectos significativos para el observador. Dentro de una red local, la latencia puede ser despreciable respecto de los tiempos de procesamiento de un cuadro, pero cuando se trabaja en un enlace a través de internet es normal que haya latencias más importantes de cientos de milisegundos, o más.

Para tratar este problema, los protocolos de comunicación suelen incluir un timestamp en la información enviada para señalar el instante en el que se generó la actualización. El receptor del mensaje puede compensar el retardo aplicando su modelo de dead reckoning para proyectar el estado del objeto al momento de su procesamiento y utilizar esa información para actualizar la imagen. Nótese que para poder realizar este ajuste, los relojes de los participantes deben estar sincronizados.

En la Figura1, se muestran los tipos de corrección que pueden realizarse sobre el modelo predicho. La curva sólida representa el estado real de una entidad en el modelo, mientras que las curvas punteadas representan los estados predichos. En el instante en que se detecta que el umbral de tolerancia se ha excedido, se envía un paquete de actualización que llega a destino luego de un tiempo de transmisión (Figura 1a).

Este estado no se utiliza directamente para realizar la corrección, sino que se extrapola un estado posterior utilizando un delta t igual a la diferencia entre el instante actual y el especificado en el time-tamp del paquete (Figura 1b). De esta manera, se evita que la entidad mostrada en pantalla realice un retroceso, lo que produciría una sensación confusa en el usuario al romper la precisión de comportamiento. Por último, para que no se aprecien saltos bruscos en el movimiento visualizado, puede suavizarse la corrección calculando un punto de convergencia al cabo de un intervalo de tiempo mayor, y luego dedicar una cantidad de cuadros para llegar a este punto (Figura 1c). Para realizar este acercamiento, puede utilizarse la fórmula de interpolación lineal definida en (1).

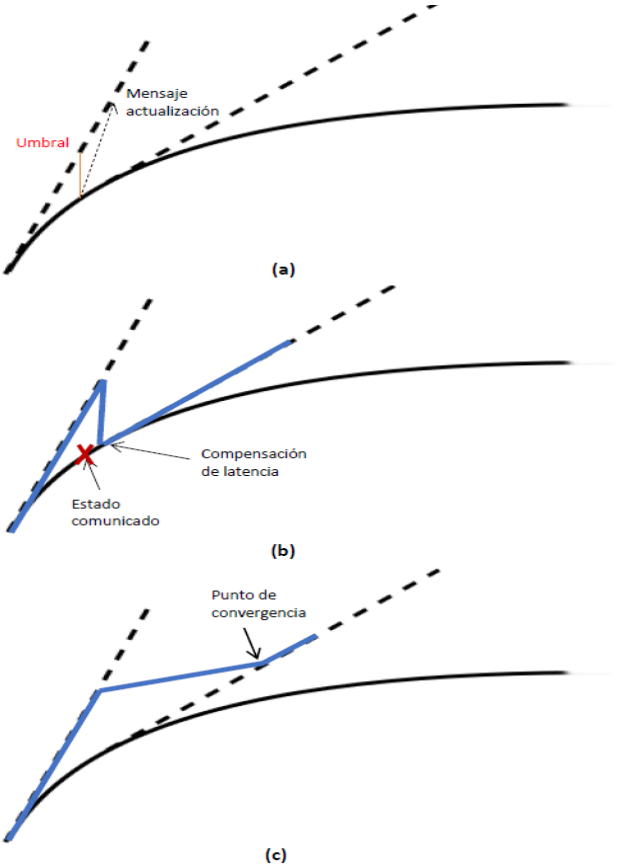

Fig. 1. Corrección del modelo de dead reckoning: envío del mensaje de actualización (a); corrección con compensación de latencia (b); corrección suavizada hacia un punto de convergencia (c).

2.4 Comunicación

La comunicación entre los hosts se lleva a cabo a través del protocolo Common Image Generator Interface (CIGI). CIGI es un estándar open source específicamente diseñado para la comunicación entre un host de simulación y un generador de imágenes (IG) [3]. Es un protocolo de empaquetamiento de datos, por lo que no depende de un medio físico de comunicación o de un protocolo de transporte específico. Toda comunicación CIGI entre un host de simulación y un IG a través de una conexión entre dos puertos de red se encapsula en una sesión del protocolo.

Se presentan modos de funcionamiento, sincrónico y asincrónico. Cuando funciona en modo sincrónico, el IG envía un paquete del tipo *Start of Frame* (SOF) al host para señalarle el inicio de cada cuadro. Este mensaje marca el ritmo con el que van a ser intercambiados los mensajes entre ambos extremos. El host de simulación responde a cada mensaje SOF con uno o varios paquetes de información que contienen el estado de movimiento de las entidades del sistema, el estado de diversos componentes u otro tipo de datos describiendo los cambios que hubo en la simulación durante el ciclo previo. Luego el host de simulación comienza su próximo ciclo computacional, mientras que el IG actualiza y dibuja la escena.

En modo asincrónico, el host de simulación no espera un paquete SOF para enviar mensajes al IG, sino que lo hace a un ritmo propio, ya sea a partir de un intervalo temporal predefinido o de la ocurrencia de algún evento. Mientras tanto, el IG mantiene su propia tasa de refresco. Antes de dibujar cada cuadro, el IG verifica que hayan llegado paquetes CIGI y, en caso afirmativo, actualiza su representación del ambiente según la información recibida. Este último modo fue utilizado en este trabajo, ya que permite la aplicación de los algoritmos de predicción tratados.

El equipo encargado de desarrollar y mantener CIGI, también provee una herramienta de software llamada CIGI Class Library (CCL) [4]. CCL se trata de una biblioteca en lenguaje C++ que implementa el protocolo tanto en el host como en el IG, proveyendo la funcionalidad necesaria para manejar el empaquetado y desempaquetado de mensajes de manera acorde.

El protocolo define una variedad de mensajes especificando su estructura. Un paquete CIGI intercambiado entre host e IG está compuesto por uno o varios mensajes. Los primeros dos bytes de cada mensaje son su encabezado; el primer byte contiene un código de operación que identifica unívocamente el tipo de mensaje del que se trata y el segundo especifica su tamaño total, en bytes.

3 Sistema de visualización

3.1 Grafos de escena

Para procesarlos datos gráficos del sistema se utilizó el concepto de grafo de escena: una estructura de datos que define las relaciones espaciales y lógicas de los elementos que componen una escena tridimensional. El grafo define tanto los datos geométricos de los vértices que componen los distintos elementos, como también transformaciones y operaciones a aplicar sobre ellos. Los resultados de una operación realizada por un nodo padre se propagan a todos sus nodos hijos, lo que permite que distintos grupos de nodos sean tratados como unidades, mejorando la eficiencia.

El concepto de grafo de escena ha sido ampliamente aplicado en software y aplicaciones gráficas modernas, como ser generadores de imágenes comerciales como VegaPrime (Presagis Inc.), Virtual Reality Scene Genera-tor (MetaVR Inc.), TacScene (Lockheed Martin Corporati-on), Vital Visual System (FlightSafety International), EP-8000 Image Generator System (Rockwell Collins), etc.; y por generadores de imágenes abiertos como OpenScene-Graph, Delta3D, etc. También son usados por motores para videojuegos como Unreal Engine o Unity, entre otros.

3.2 Representación del modelo

Para llevar a cabo las técnicas de generación de gráficos propuestas, se utilizó la herramienta OpenSceneGraph (OSG): un framework de gráficos tridimensionales open source, de alta performance, escrito íntegramente en C++ y que trabaja por encima de la API de OpenGL.

El grafo de escena del entorno simulado está compuesto por los modelos 3D que representan a cada una de las entidades. Estos modelos son típicamente creados en editores de diseño externos, e importados a OSG a través de un formato estándar de intercambio, lo que genera subgrafos de escena de complejidad variable. Cada uno de estos subgrafos está subordinado a una matriz de transformación, que define la posición y orientación de la entidad dentro de la escena. Esta matriz es actualizada mediante la aplicación de una función de callback: funciones creadas por el usuario que se asignan al nodo y se ejecutan encada cuadro durante su procesamiento (Figura 2).

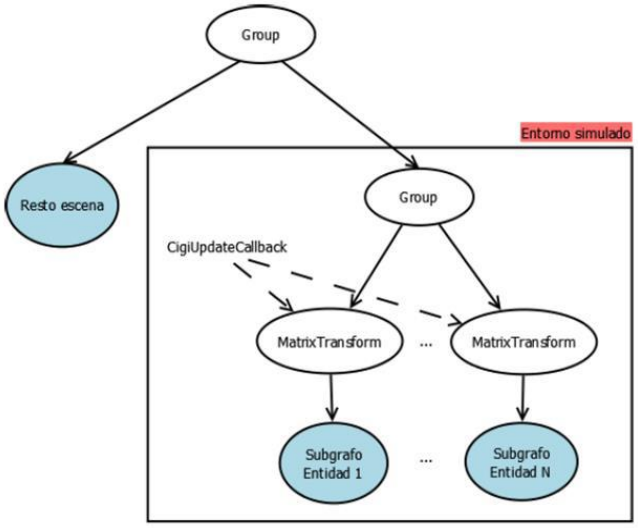

Fig. 2. Estructura del grafo de escena que contiene la representación del entorno simulado.

El entorno simulado es sólo una parte del grafo de la escena. Además puede contener cualquier elemento adicional que aumente la riqueza de la visualización (cámaras, fuentes de luz, fondos, otros modelos estáticos, etc.) que dependerán de las características del sistema que se esté simulando.

3.3 Actualización de la escena

Para actualizar el estado de movimiento de las entidades en pantalla según los paquetes CIGI entrantes, se desarrolló la clase de actualización CigiUpdateCallback. Esta clase define una función, según la metodología de OSG, que aplica una modificación sobre el nodo de transformación en tiempo de ejecución.

Por otro lado, implementa el algoritmo de dead reckoning, siendo la entidad visualizada aquella correspondiente a la copia predicha del modelo, mientras que el estado original se monitorea de manera pasiva hasta que se encuentre un cambio. Al momento de actualizar la matriz de transformación, pueden darse dos variantes:

1. No han llegado paquetes de actualización a través de la red. Lo que implica que el estado de movimiento de la entidad debe predecirse extrapolando a partir del último estado conocido.
2. El modelo ha sido actualizado a partir de un paquete entrante. En este caso, se debe calcular un punto de convergencia entre el modelo original y la copia y dedicar los próximos cuadros a realizar una corrección suave.

Para realizar la predicción, se guarda el instante temporal en el cual el callback se ha llamado por última vez (esto es, durante el procesamiento del cuadro anterior), de manera de poder calcular el paso temporal necesario para realizar la extrapolación.

También se guarda el último estado de movimiento del modelo, para poder detectar si se han producido cambios en el mismo.

4 Sistema de simulación

4.1 Concepto

En este sistema, se plantea un modelo dinámico de un sistema físico de interés para su posterior simulación. El modelado se realizó utilizando Modelica, que se trata de un lenguaje de programación orientado al modelado de sistemas dinámicos [5], a través de OpenModelica, un entorno open source de modelado y simulación basado en este lenguaje.

La funcionalidad de control que implica desarrollar las técnicas de predicción necesarias y empaquetar el estado de la simulación en mensajes CIGI se desarrolló utilizando un lenguaje de propósito más general, en este caso, C++. La adición al entorno de Modelica fue la de un módulo de transferencia entre procesos que permite la comunicación con sistemas externos a través de sockets. De esta manera, la tarea del sistema físico funcionando en OpenModelica es la de resolver las ecuaciones que componen el modelo y comunicar a intervalos regulares el estado del mismo a un sistema de control. Este sistema, a su vez, procesa ese estado y, si corresponde según el algoritmo de predicción que se esté utilizando, lo empaqueta en mensajes CIGI y lo envía a través de la red. Para eso, mantiene una sesión CIGI con el generador de imágenes remoto (Figura3).

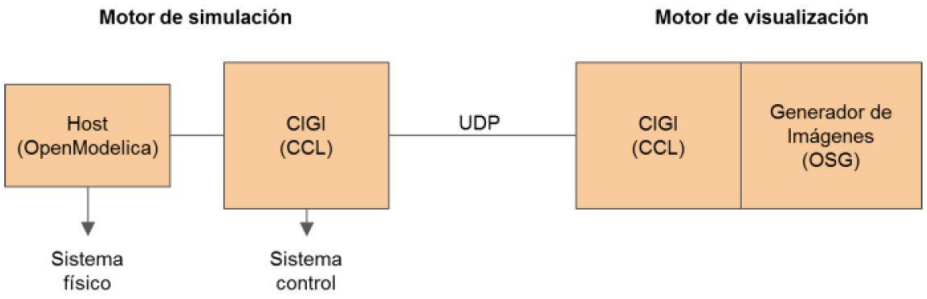

Fig. 3. Esquema de la solución propuesta. El motor de simulación compuesto por un sistema físico y un sistema de control se comunica con el motor de visualización utilizando el protocolo UDP.

4.2 Paquetes CIGI

El paquete de actualización enviado a través de la red contiene información necesaria acerca del ejercicio de simulación, así como la descripción del estado de movimiento de las entidades que en él participan (Figura4).
Está compuesto por los siguientes mensajes, según el protocolo CIGI:

1. Información de control del ejercicio, utilizando el mensaje *IGControl*, con la información propia del protocolo e incluyendo un time-tamp a utilizarse para aplicar técnicas de compensación de latencia.
2. El estado de movimiento de las entidades simuladas, utilizando un mensaje *Entity Control* por cada una de ellas. Esto incluye las tres coordenadas de posición y los tres ángulos que definen su orientación en el espacio.
3. La velocidad de las entidades simuladas, utilizando el mensaje *Rate Control*.
4. La aceleración de las entidades simuladas, para realizar predicciones de segundo orden, utilizando el mensaje *Trajectory Definition*.

5 Aplicación

El software y las técnicas desarrolladas se utilizaron para simular y visualizar un caso particular de estudio: el movimiento de una formación ferroviaria a través de un tramo de vía (Figura5). A partir de los resultados obtenidos, se analizó el impacto de los métodos de predicción y compensación de latencia estudiados, ejecutando el sistema en un entorno con una latencia agregada.

La trayectoria de la vía se modeló a partir de curvas de Bézier cúbicas. Estas curvas proveen un mecanismo para generar tramos de curva suaves a partir de una serie de puntos de control. Los puntos de la curva se calculan a partir de un parámetro u que varía entre 0 y 1, según la ecuación:

$$\mathrm{x(u)} = \sum_{i=0}^{n} B_i^n(u)P \qquad (2)$$

$$B_i^n(u) = \binom{n}{i}(1-u)^{n-1}u^i$$

La forma cúbica se obtiene con *n* igual a 3 en la ecuación (2) y se define a partir de cuatro puntos de control.

Si bien las herramientas desarrolladas en este trabajo nos permiten trabajar con movimientos que tienen seis grados de libertad (tres coordenadas de posición y tres ángulos de orientación), el modelo aquí planteado sólo presenta uno: la posición actual en la vía. El movimiento está definido a partir dela variación del parámetro u respecto del tiempo de simulación. Si tanto el simulador como el IG conocen la geometría de la vía de antemano, puede intercambiarse solamente el valor del parámetro y mejorarla precisión en el sistema de visualización.

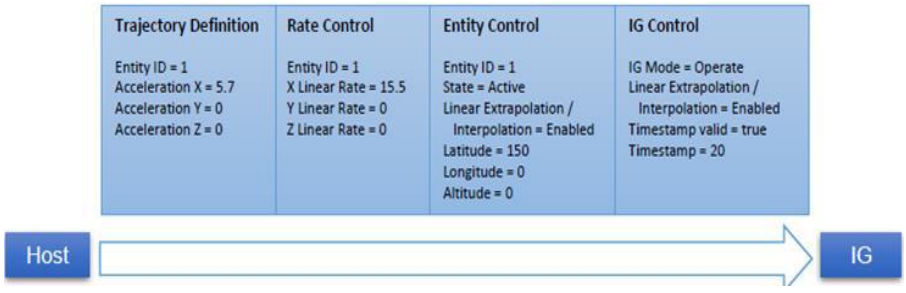

Fig. 4. Ejemplo de paquete CIGI enviado desde el Host al IG, utilizando dead reckoning. Comunica el estado de movimiento de una entidad del modelo físico e información de control del ejercicio. Se muestran los campos más importantes de cada mensaje.

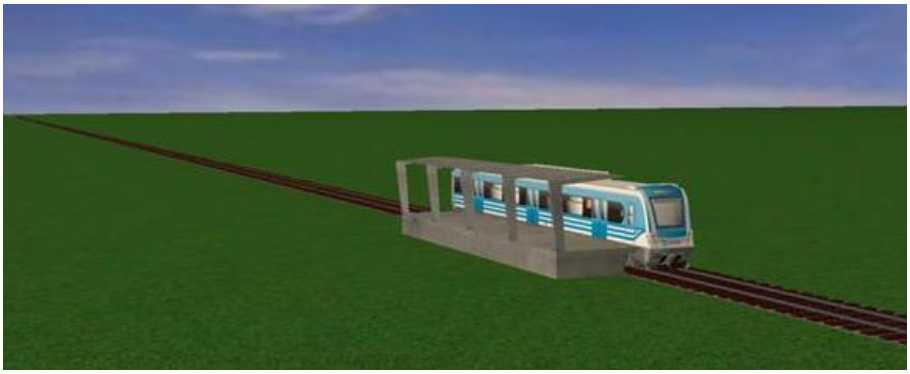

Fig. 5. Captura de pantalla de la escena visualizada para una simulación ferroviaria

En la Figura 6 se muestra la trayectoria que recorre la entidad simulada en OpenModelica, junto con la variación del parámetro u respecto del tiempo que define su movimiento. La comparación de la progresión del parámetro en el sistema físico y la predicha por el IG se muestra en la Figura 7. Cuando no se usa dead reckoning (Figura 7a), pueden verse los efectos adversos que se deben corregir. En primer lugar, el tiempo de latencia de la comunicación hace que el movimiento esté atrasado respecto del original, lo que produce un corrimiento entre ambas trayectorias. En segundo lugar, el hecho de que la pantalla se actualiza a mayor velocidad que aquella a la que se generan estados produce una trayectoria escalonada, donde el usuario advierte que la entidad se mueve a pequeños saltos.

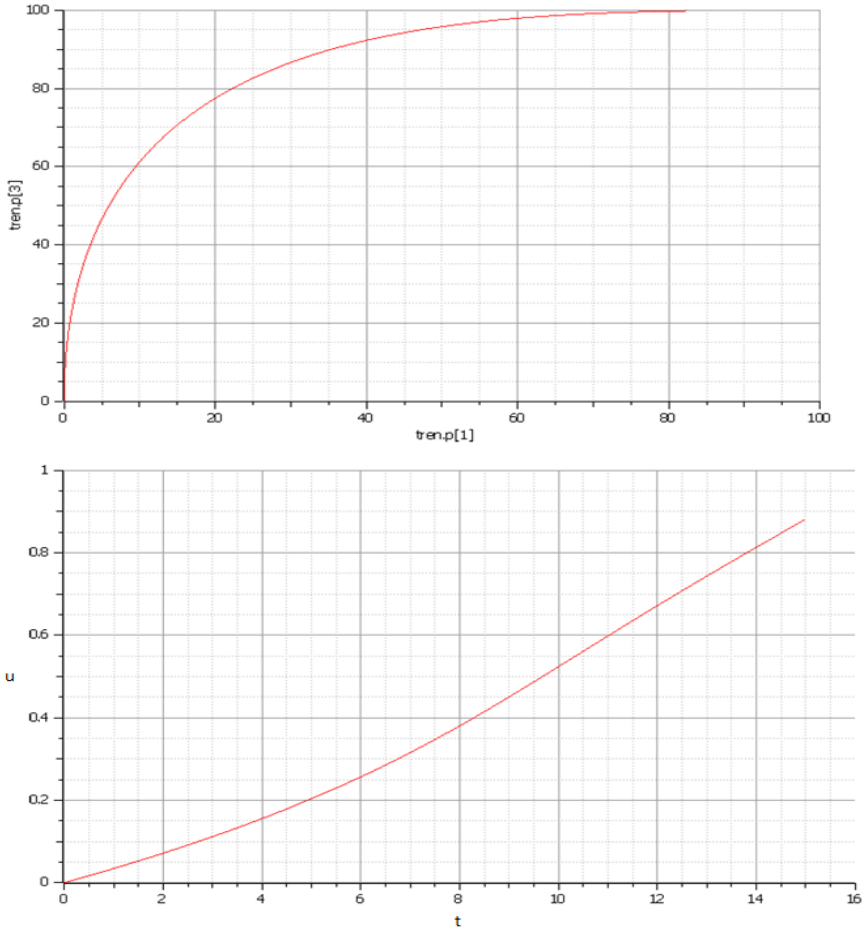

Fig. 6. Trayectoria definida por los puntos P0=(0,0,0), P1=(0,0,100), P2=(50,0,100), P3=(100,0,100) (arriba). Progresión del parámetro u respecto del tiempo (abajo). Gráficos generados por OpenModelica.

Cuando se usa dead reckoning y compensación de latencia (Figura 7b), puede observarse cómo se mitigan estos efectos. Existe una corrección inicial, que es cuando el IG advierte que la entidad se está moviendo y, a partir de ese momento, el parámetro de movimiento se predice con un alto grado de precisión. Pueden verse varios puntos en los que el parámetro debe ser corregido por haberse desviado del original en una magnitud mayor al umbral de tolerancia, que en este caso es del 1%.

Por último, debe ser analizado el factor de utilización de la red. Cuando no se usa dead reckoning, el simulador debe enviar todas las actualizaciones de estado producidas, que en este caso son diez por segundo. Cuando se usa dead reckoning, se envían el estado inicial y final y las correcciones intermedias, una cantidad considerablemente menor.

El impacto de las correcciones en la visualización es despreciable, ya que cumple con los tres requisitos definidos: precisión de posición, precisión de comportamiento y renderizado suave. En el caso de un simulador ferroviario, los movimientos se tratan de trayectorias suaves, por lo que se puede definir un umbral de tolerancia bajo y lograr tasas de actualización muy pequeñas. Si se aumenta el umbral de tolerancia al 5% (Figura 7c), puede verse cómo disminuye la cantidad de mensajes de actualización a cambio de una menor precisión en el comportamiento visualizado. La definición de este parámetro es un elemento de diseño cuya eficacia depende de las características propias de cada ejercicio de simulación.

6 Trabajos futuros

Los trabajos siguientes se enfocarán en aplicar este desarrollo a un sistema de simulación completo, apto para el uso de un usuario en ejercicios de entrenamiento. Para alcanzar esta meta, se ampliarán las funcionalidades en los dos frentes: el modelo físico y el generador de imágenes.

Respecto del modelo físico, se incluirá el modelado de distintos elementos que sean de interés para el entrenamiento del conductor, de manera de poder reproducir los mecanismos que entran en funcionamiento en una cabina de manejo. Estos mecanismos varían respecto del objetivo del ejercicio particular que se desee desarrollar.

Algunos de ellos son: tracción, frenado, cruces, señalización o comunicaciones.
En cuanto al generador de imágenes, debe incluirse una representación de estos mecanismos. Se desarrollará una vista de cámara dentro de la cabina, modelando gráficamente los controles que se encuentran en ella, de manera de realimentar el modelo interactuando con ellos. También se trabajará en la representación del terreno y la vía, en búsqueda de lograr escenas más realistas y eficientes. Esto implica el relevamiento del terreno sobre el que se quiere entrenar y el manejo de técnicas visuales de nivel de detalle y paginado de memoria para representar terrenos de gran extensión.

Por otro lado, el sistema de simulación será localizado en un ambiente propicio para llevar a cabo los ejercicios. El mismo puede ser desde un puesto de control con una interfaz gráfica que presente los controles, hasta una cabina de simulación con movimiento integrada al funcionamiento del motor físico.

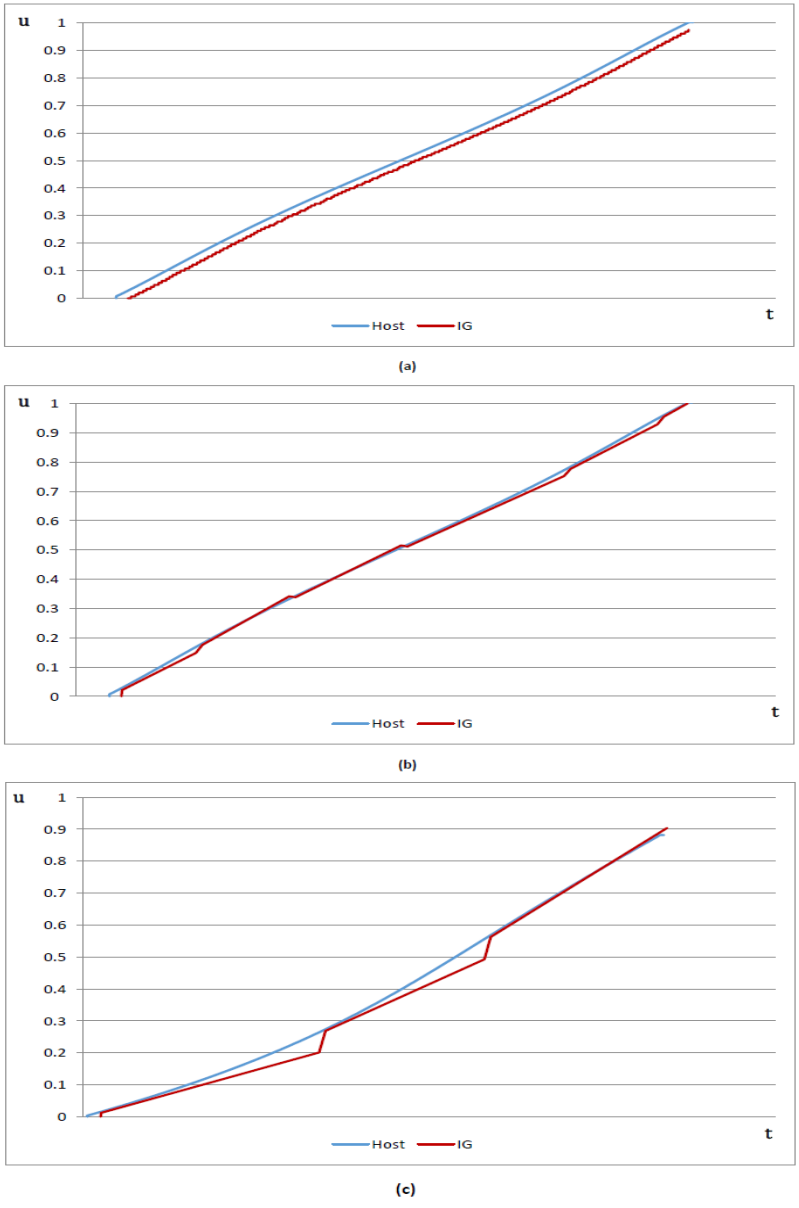

Fig. 7. Comparación de progresión del parámetro u respecto del tiempo en el Host y en el IG. La comunicación presenta una latencia de 300 ms: (a) sin dead reckoning; (b) con dead reckoning, compensación de latencia y tolerancia del 1%; (c) con dead reckoning, compensación de latencia y tolerancia del 5%.

Es el propósito final, liberar estas herramientas en la forma de una biblioteca abierta creada sobre OpenSceneGraph para visualización de entornos virtuales controlable

mediante el protocolo CIGI, de manera que la comunidad pueda ampliarla y utilizada para la simulación de otros tipos de modelos físicos, así como la inclusión de otras técnicas de visualización.

7 Agradecimientos

Este trabajo se desarrolló con financiamiento de los proyectos:

1. "Simulador de entrenamiento para conductores ferroviarios" 32-64-007 de la convocatoria "Universidad y transporte argentino" de la Secretaría de Políticas Universitarias (SPU) del Ministerio de Educación y llevado a cabo por la Universidad de Buenos Aires y la Universidad Nacional de Rosario.
2. "Simuladores de entrenamiento con movimiento para conducción de vehículos" PIDDEF 03/14 SIMMOV (MINDEF), llevado a cabo CITEDEF.

8 Referencias

1. Singhal, S. y Cheriton, D.: "Using a Position History-Based Protocol for Distributed Object Visualization," Department of Computer Science, Stanford University, 1994.
2. IEEE Computer Society, "IEEE Standard for Distributed Interactive Simulation – Application Protocols", 2012.
3. Durham, L. y Bill, P.: "Interface Control Document for the Common Image Generator Interface (CIGI)", http://cigi.sourceforge.net/specification.php, 2008.
4. "Cigi Tools: Cigi Class Library", http://cigi.sourceforge.net/product_ccl.php.
5. "Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification. Version 3.3 Revision 1", Modelica Association, 2014.