CAMBIO DE PARADIGMA EN EL CONTROL DE APLICACIONES GRÁFICAS EN TIEMPO REAL

TRABAJO FIN DE MÁSTER 2010-2012
Con orientación Profesional

Realizado por: Vicente Broseta
vbroseta@dsic.upv.es

Dirigido por: Ramón Mollá
rmolla@dsic.upv.es

Valencia, 2012

Perteneciente al Máster
"Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital"
del DSIC (Departamento de Sistemas Informáticos y Computación)
en la UPV (Universidad Politécnica de Valencia)

Dedicatoria

A mis padres, porque cada día los quiero más
A mis hermanos, que los quiero un montón
A Marisa y Tere, por quererme como soy

Agradecimientos

Muy especialmente dar mil gracias a Ramón por su paciencia, su dedicación, su interés, su intención, sus conocimientos y su buena voluntad

A Toni Barella, por su gran ayuda y buena voluntad en ayudarme a empezar

A mis compañeros de Máster por tantas conversaciones fructíferas en especial a Joan, Flavio y Juan Fer. Juan Fer si no fuera por ti …

A Guisela por animarme a iniciar esta aventura

La fe es la fuente de la realidad, porque es la vida; creer es crear.
(Miguel de Unamuno)

El científico no tiene por objeto un resultado inmediato. Él no espera que sus ideas avanzadas sean fácilmente aceptadas. Su deber es sentar las bases para aquellos que están por venir, y señalar el camino
(Nikola Tesla)

RESUMEN

Tradicionalmente existen dos grandes áreas que se han desarrollado de forma independiente en el campo de la simulación. Por una parte esta la Simulación en Tiempo Real típicamente utilizada en video-juegos y por otra parte esta la Simulación Discreta típicamente utilizada en simulación científica basada en teoría de colas. En este trabajo se pretende unir estas dos áreas aplicando la tecnología de la Simulación Discreta a el control interno de las Aplicaciones Gráficas en Tiempo Real (AGTR). Esta transferencia tecnológica supone un cambio de paradigma en el desarrollo de las AGTR. Este cambio de paradigma permite una simulación mucho más precisa debido al control más eficiente realizado sobre la potencia de calculo mejorando así la productividad de la CPU. Cada aspecto de cada objeto de la AGTR puede ser ejecutado de forma independiente y de esta forma satisfacer sus propios requerimientos. Por lo tanto, la utilización de la CPU solo se hace cuando es estrictamente necesario liberando potencia de calculo para otros propósitos como mejorar, entre otras cosas, la calidad de las aplicaciones, la IA de las aplicaciones, la posibilidad de migrar aplicaciones a dispositivos con menos recursos como dispositivos móviles, dispositivos antiguos o simplemente ahorrar energía para incrementar el tiempo de juego o aumentar el tiempo de las baterías.

Este es un proyecto con orientación profesional, su objetivo es utilizar una tecnología, ya existente y recientemente mejorada, en el desarrollo de un Benchmark mediante el cual poder hacer una serie de mediciones y un Demostrador que facilite la transferencia de la tecnología desarrollada a el mundo empresarial.

Tabla de Contenidos

Índice de Figuras

1 Introducción

1.1 Simulación en Tiempo Real

A lo largo del tiempo se ha trabajado de forma separada en dos tipos de simulación: Simulación con Tiempo Real y Simulación sin Tiempo Real también llamada Simulación Discreta. Las Aplicaciones Gráficas en Tiempo Real (AGTR) son un ejemplo típico de Simulaciones en Tiempo Real como por ejemplo son los Video-juegos, la Realidad Virtual, la Realidad Aumentada, los Simuladores 3D, los Mundos Virtuales, …

La Calidad de Servicio que deben de proporcionar las AGTR es un factor muy importante que debe de cumplirse para que la sensación del usuario sea satisfactoria. La Calidad de Servicio incluye un amplio abanico de elementos que forman parte de la AGTR como pueden ser:

- Latencia de Respuesta: Tiempo que se tarda desde que el usuario, ya sea a través de un periférico o a través de internet, realiza una acción hasta la ejecución de la tarea correspondiente.

- Calidad de la IA: Dependiendo de la profundidad de los arboles de busqueda la calidad de los resultados puede ser muy importante.

- Nivel de Detalle Gráfico: La calidad de los gráficos dependen del tamaño de las mallas utilizadas.

- Frecuencia de Muestreo: La frecuencia a la que los objetos de la AGTR deben de calcular la simulación influye en una buena calidad en la detección de colisiones, en la continuidad de los movimientos de los objetos, en que las acciones se realicen en el instante preciso, …

A lo largo de este trabajo se va ha hacer mucho hincapie en este último elemento que forma parte de la Calidad de Servicio, la Frecuencia de Muestreo. A lo largo de esta

Tesina, a esta Frecuencia de Muestreo se le va a aplicar la nomenclatura QoS.

Una de las principales características de las AGTR es conseguir la suficiente velocidad para realizar los cálculos de actualización de todos los objetos que pertenecen al espacio simulado o lo que es lo mismo, la QoS. En las AGTR el objetivo principal es que se cumpla la QoS de todos los objetos del espacio simulado y al mismo tiempo realizar el proceso de visualización sin perder calidad en la sensación de tiempo real del usuario. La forma habitual de gestionar estas tareas es mediante un bucle infinito que entrega los recursos (CPU, GPU, Red, Memoria, …) a cada tarea de forma sistemática y periódica tantas veces como sea posible. Por este motivo, podemos decir que en general para las Simulaciones en Tiempo Real la velocidad prima sobre la calidad.

La evolución de la complejidad de las AGTR siempre a estado ligada a la mejora de la velocidad de las CPU, primero con las CPU mono-núcleo, seguidas de las CPU multinúcleo y multihebra y por ultimo la aparición de las GPGPU con cientos de núcleos generalistas en los cuales se pueden ejecutar estos bucles directamente [2] [3] permitiendo el equilibrio entre CPU y GPGPU [4]. La utilización de cientos de procesadores gráficos trabajando en paralelo ha acaparado toda la atención científica estirando el modelo de ejecución tradicional hasta el máximo. Hoy día, en general las lineas de investigación están enfocadas hacia diferentes opciones como puedan ser la sincronización de hebras, distribución de bucles de trabajo, criterios para mejorar la distribución de tareas entre procesadores o hebras, dead-locks ,técnicas de recuperación frente a interbloqueos, …

Curiosamente el núcleo de control de las AGTR siempre ha estado bajo el mismo paradigma de programación, este modelo ha sido suficientemente estudiado en el pasado [1]. Por mucho que se dividan las AGTR en pequeños subprocesos, cada uno de los procesos en ejecución sigue el diseño del viejo paradigma. El viejo paradigma consiste fundamentalmente en que cualquier AGTR mono-hilo o cada hilo de una AGTR multi-hilo primero realiza el proceso de Simulación y seguidamente el proceso de Visualización. Todos los objetos pertenecientes al mundo virtual tienen que evolucionar en función de unos eventos de usuario que bien pueden venir de la red en un entorno distribuido o bien de periféricos externos conectados al computador (fase de Simulación) y de forma inmediata debe realizarse la sincronización de los render visual,

auditivo y háptico, a través de los cuales se reflejaran todos esos cambios (fase de Visualización). Como se ha comentado anteriormente, estas dos fases se deben ejecutar a la suficiente velocidad para garantizar la sensación de tiempo real en el usuario.

1.2 Simulación Discreta

Por otra parte, esta la Simulación Discreta. Ejemplos típicos de Simulación Discreta pueden ser los empleados en diferentes ámbitos como el:

- Social: Movimientos migratorios, modelos de evacuación, redes sociales, …
- Industrial: Modelado de cadenas de suministro y fabricación, redes de abastecimiento de agua, gas o electricidad, …
- Comunicaciones: Redes de telefonía, flujos de internet, modelado de carga, tráfico aéreo, carreteras, ferroviario, …
- Económico: Análisis de mercados, simulación de mercados financieros, hábitos de compras, …
- Clima: Predicción del tiempo, predicción del clima global, …
- Biología sintética [5]: Simulación celular.

Los objetivos de algunas de estas aplicaciones suelen ser, entre otros, la ayuda de toma de decisiones asistida por computador (simulación industrial) o valoraciones de modelos predictivos (simulación científica). Para este tipo de simulaciones la calidad prima sobre la velocidad, la precisión de los cálculos y las mediciones realizadas tienen mucha más importancia que la velocidad de realizarlos. Normalmente se desarrollan aplicaciones ad-hoc debido a la complejidad y especificidad de los sistemas que se quieren simular. Las tecnologías utilizadas para la Simulación Discreta no suelen ser soluciones estándar o paquetes de software generalizados que puedan aplicarse de forma universal a cualquier tipo de simulación. No obstante si que existen lenguajes de programación orientados a la simulación [6]. Últimamente se está empleando mucho la simulación y modelado basado en agentes (ABMS). Es una técnica que surgió en los años 90 en un entorno multidisciplinar está muy relacionada con la gestión de eventos discretos [7] y para la que existen actualmente muchas herramientas y entornos de desarrollo [8].

1.3 Motivación

En el año 2004 se presenta una tecnología que aprovecha las ventajas de la precisión en la Simulación Discreta aplicando estas a la Simulación en Tiempo Real. Esta tecnología propone cambiar el paradigma continuo y acoplado de la simulación de las AGTR a un paradigma discreto desacoplado. Este nuevo paradigma permite solucionar los problemas del paradigma continuo acoplado. Los objetos definen su propia QoS independientemente del resto del sistema, incluso se permite definir diferentes QoS para diferentes aspectos del propio objeto. En el sistema discreto desacoplado cada objeto consume únicamente la potencia de cálculo estrictamente necesaria para llevar a cabo su simulación y cumplir con su QoS definida. Por ello, la potencia de cálculo del sistema se reparte entre los objetos en función de sus necesidades [13].

Ademas, el sistema puede adaptarse dinamicamente durante la ejecución redefiniendo la QoS tanto de los objetos como de cualquier aspecto simulado de cada objeto en función de las condiciones de la ejecución del sistema. Los objetos pueden degradar o mejorar su comportamiento durante periodos de la ejecución para evitar los colapsos del sistema o mejorar el comportamiento del sistema [13].

Cuando se menciona colapso del sistema se hace refencia a cuando el sistema no es capaz de simular y visualizar correctamente en un ciclo de refresco de pantalla. El colapso del sistema tiene efectos diferentes en el sistema continuo y en el discreto. El sistema continuo colapsado tiene un funcionamiento incorrecto (colisiones no detectadas, trayectorias incorrectas,...). El comportamiento incorrecto se acentúa cuando aumenta el factor de colapso. El sistema discreto en cambio simula el sistema correctamente independientemente del colapso del sistema. Para conseguir una simulación correcta sacrifica la ejecución del sistema en tiempo real. El sistema discreto colapsado consume un tiempo real en ejecutar la simulación mayor que el tiempo de simulación del sistema. La diferencia entre estos dos tiempos será mayor cuanto mayor sea el factor de colapso. Ante el colapso del sistema, el sistema discreto degrada la QoS de los objetos, pero el sistema discreto la mantiene [13].

1.4 Objetivo

El objetivo de este trabajo es aplicar esta tecnología para transformar una AGTR, que ya este implementada bajo el paradigma continuo acoplado, al nuevo paradigma discreto desacoplado y hacer un análisis de las ventajas que puede ofrecer este nuevo paradigma. Además, se pretende comprobar la validación de los datos obtenidos en [13] a pesar del desarrollo tecnológico experimentado en los últimos 8 años, tanto en hardware como en las herramientas de desarrollo. Así mismo, para que la comparativa sea completa, se ha realizado otra aplicación completamente nueva realizada "from scratch" que además se implementa con Glut/OpenGl que es una tecnología alternativa frente a las obsoletas usadas en la tesis [13]: núcleo continuo acoplado de videojuegos Fly3D vs GLUT/Open GL.

Otro objetivo de esta Tesina es la realización de un tutorial que sirva de guía para futuros programadores que utilicen Glut/OpenGl y quieran aplicar esta tecnología.

1.5 Aportaciones

Las aportaciones realizadas en esta Tesina son:

- Ayuda a la divulgación y entendimiento de la utilización del Paradigma Discreto Desacoplado en las AGTR.
- Comparativa de resultados para entender algunas de las muchas ventajas que implica utilizar la tecnología RT-DESK.
- Validación de los resultados de la Tesis de Inmaculada García [13].
- Aplicación tipo Demo para facilitar la transferencia tecnológica al mundo empresarial.
- Tutorial para futuros programadores que puedan estar interesados en desarrollar con Glut/OpenGl y RT-DESK.

1.6 Contenido

La estructura de esta Tesina se divide en los siguientes capítulos:

1.6.1 Estado del Arte

Este capítulo se divide en los siguientes apartados:

1. Visualización y Simulación. Realidad Virtual.
2. Simulación de Eventos Discretos
3. Conclusiones
4. Crítica
5. Propuesta de Mejora

Esta Tesina se centra en el estudio y aplicación para transformar las AGTR desarrolladas bajo el típico Paradigma Continuo Acoplado en AGTR que utilicen el nuevo Paradigma Discreto Desacoplado.

Para ello se parte de un estudio previo que muestra el problema generado por el acoplamiento de las fases de Visualización y Simulación en las AGTR y como en concreto las aplicaciones de Realidad Virtual han solucionado el problema del acoplamiento paralelizando estas dos fases.

Luego se ha realizado un breve estudio sobre el funcionamiento de los Simuladores de Eventos Discretos para entender su filosofía de funcionamiento y así poder aplicar de forma correcta la tecnología RT-DESK cuyo núcleo esta formado por un Simulador de Eventos Discretos en Tiempo Real.

Posteriormente se llega a unas conclusiones y se procede a realizar una crítica donde se indican las desventajas del Modelo Continuo Acoplado y a continuación una propuesta de mejora donde se indican como el Modelo Discreto Desacoplado soluciona estas desventajas y genera nuevas posibilidades para la mejora de las AGTR.

1.6.2 RT-DESK

En este capítulo se presenta la tecnología RT-DESK. Se explica brevemente como se ha sido su evolución desde su origen DESK hasta lo que es hoy en día y luego se explica que es y como funciona.

1.6.3 Benchmark

En este capítulo se explica en el Benchmark realizado. Primero se indica el objetivo y el entorno de software y hardware donde se realiza y luego se explica el desarrollo, la configuración y el funcionamiento de la aplicación implementada. Por ultimo se realiza una explicación sobre como se realizan las tomas de tiempo y como se generan los datos de salida para su posterior análisis.

1.6.4 Resultados

En este capítulo se realiza un análisis de los resultados obtenidos después de diferentes ejecuciones con distintas configuraciones.

1.6.5 Conclusiones

En este capítulo se indican las conclusiones obtenidas después de analizar los resultados obtenidos y observar el funcionamiento de la aplicación tanto en Modo Continuo Acoplado como en Modo Discreto Desacoplado.

1.6.6 Trabajos Futuros

En este capítulo se plantean los trabajos que son inmediatamente necesarios para completar el funcionamiento óptimo de RT-DESK. También se plantean las posibles lineas de trabajo que se pueden seguir para conseguir que RT-DESK sea más potente y se adapte mejor a cualquier tipo de software y hardware en la actualidad.

1.6.7 Anexos

La Tesina contiene los siguientes Anexos:

- Anexo A: Muestra el código de inicialización de la aplicación como ayuda a programadores que quieran utilizar RT-DESK. En el se muestra tanto la inicialización de

RT-DESK como el lanzamiento de los primeros eventos para que RT-DESK tome el control de la aplicación.

- Anexo B: En este Anexo se muestra un ejemplo de las salidas csv que genera la aplicación y se explican el significado de los valores que contiene.

- Anexo C: En este Anexo se expone un breve tutorial que sirva como guía a un futuro programador que quiera integrar RT-DESK en una aplicación que utilice Glut/OpenGl.

- Anexo D: Con la finalidad de poder facilitar el objetivo de transferir la tecnología RT-DESK al mundo empresarial, se ha desarrollado una Demo y en este capítulo se explica el cual es funcionamiento y como se utiliza.

2 Estado del Arte

2.1 Visualización y Simulación. Realidad Virtual.

En los inicios de las AGTR se trabajaba con un bucle principal en el que la fase de simulación estaba acoplada con la fase de visualización. Esto quiere decir que cada vez que se realizaba un cambio de alguno de los objetos del mundo virtual se procedía a una actualización de la fase de visualización.

Este acoplamiento se puede producir de dos formas diferentes. Si se utiliza un grafo de escena el acoplamiento se puede realizar en cada objeto de manera que se aprovecha el recorrido del grafo para simular y visualizar al mismo tiempo cada uno de los objetos o primero se simulan y luego se visualizan todos objetos a la vez. Si no se utiliza grafo de escena el acoplamiento se realiza al primero simular y luego visualizar.

Si se consigue desacoplar estas dos fases, la frecuencia de visualización es más rápida, incluso si las escenas son más complejas [22]. El desacoplo incrementa el rendimiento del sistema [23].

El desacoplo es muy utilizado en las aplicaciones de Realidad Virtual. Las aplicaciones de Realidad Virtual son un tipo de AGTR y tienen características como la interacción con el usuario, trabajan en tiempo real, utilizan detección de colisiones, utilizan algoritmos de visualización y tienen objetos cuyo comportamiento utiliza técnicas de IA.

Algunos ejemplos de sistemas de realidad virtual que utilizan técnicas de desacoplamiento son:

- Cognitive Coprocessor Arquitecture, 1989. [24].
- Interface de Dialogo, 1991. [25].
- MR ToolKit, 1992. [14]
- Alice&Diver, 1994. [15]
- Bridge, 1995. [16].

El desacoplo de las fases de Simulación y Visualización permite incrementar la precisión y velocidad del sistema, ademas permite que la velocidad de los procesos de sistema sea independiente [26].

2.2 Simulación de Eventos Discretos

La Teoría de Sistemas define un sistema como cualquier entidad real o artificial, que puede estar compuesta por entidades más simples con relaciones espaciales o temporales. La interacción entre las entidades del sistema define sus características y se traduce en la evolución del sistema hacia un objetivo concreto [27].

Un evento o suceso es el cambio de estado de una entidad del sistema, una ocurrencia instantánea que altera el estado del sistema [21]. El estado de una entidad lo define el valor de sus atributos. El estado del sistema lo define el conjunto de todos los estados de todos los objetos y entidades que forman el sistema [13].

Según [28], los sistemas pueden ser clasificados de diversas formas. Para el objetivo de este trabajo se va a utilizar la clasificación en función del modo en que evoluciona el tiempo cuando se produce un evento dentro del sistema. Según esta clasificación existen tres tipos de sistemas, los Continuos, los Discretos y los Híbridos.

- Sistemas Continuos: El flujo de información evoluciona de forma continua al igual que los sistemas propios de la naturaleza.

- Sistemas Discretos: El flujo de información evoluciona a intervalos de tiempo de manera que los procesos del sistema solo se ejecutan en estos intervalos de tiempo. Algunos ejemplos de este tipo de sistemas son las lineas de producción, control de tráfico, redes de ordenadores, ...

- Sistemas Híbridos: En estos sistemas conviven procesos que se comportan de forma continua y procesos que se comportan de forma discreta [29].

La simulación esta definida por [30] como la imitación del comportamiento

dinámico de un sistema con el fin de llegar a conclusiones aplicables al mundo real. La simulación permite variar el comportamiento del sistema y analizar los resultados.

La simulación es un método experimental que en lugar de experimentar con un sistema real se experimenta con un modelo de simulación que se desarrolla explícitamente para el sistema real en cuestión. Un modelo es una representación abstracta de un sistema [31].

La Simulación de Eventos Discretos se utiliza para modelos de sistemas donde los cambios de estado pueden representarse mediante la ocurrencia de una serie de sucesos en quantos de tiempo discretos [21] [32]. Los cambios en el sistema ocurren en el momento del evento, por tanto el tiempo del sistema avanza con la ocurrencia de eventos.

2.3 Conclusiones

La simulación de eventos discretos es una técnica de simulación muy importante hoy día debido a sus características de poder realizar estudios y observaciones muy detallados en ámbitos tan importantes como el Social, Industrial, Comunicaciones, Científico y Económico. Ya en el año 1999 se realizo el intento de unificar el campo de la simulación discreta con el entorno de las aplicaciones gráficas de forma que la salida de la simulación fuese una secuencia gráfica [33].

En las AGTR la simulación y la visualización están fuertemente acopladas. Este acoplamiento obliga a que para realizar una simulación se tenga forzosamente que realizar una visualización aunque esta no vaya a ser mostrada. Como excepción, *OpenGl Performer* desacopla estas fases [13].

Las aplicaciones de realidad virtual tienen un componente de simulación muy fuerte. Las exigencias de velocidad de interacción con el usuario son elevadas. Por ello, la carga de visualización suele minimizarse con escenas sencillas. La carga de visualización es también muy elevada, aunque se empleen técnicas para disminuir esa carga, como texturas pequeñas, baja poligonalización o pocas luces. La complejidad de estos sistemas es tan elevada, que se tiende a distribuir o paralelizar la aplicación para

poder abordarla.

2.4 Crítica

Dentro del campo de la Simulación en Tiempo Real el objeto de estudio son las AGTR. El núcleo de gestión de las AGTR sigue un paradigma de ejecución continuo y acoplado. Es decir, la frecuencia a la que se deben realizar los cálculos de actualización de los objetos dentro del mundo virtual (proceso de simulación) afecta a la frecuencia con la que se debe renderizar cada escena de este (proceso de renderizado) ver Figura 1.

Para garantizar que no se pierde ninguna actualización de ningún objeto en el mundo virtual o lo que es lo mismo, que se cumple la QoS de todos los objetos, el sistema debe de ir a la máxima velocidad posible evaluando constantemente si algún objeto sufre algún cambio. Si un objeto evoluciona muy rápido obliga a ejecutar el proceso de renderizado a la misma velocidad provocando una frecuencia de render muy superior a la necesaria para dar una buena calidad al usuario.

Dado que el proceso de renderizado tiene un consumo de recursos que generalmente es bastante elevado se genera una limitación extra en el tiempo dedicado al proceso de simulación. Este es el problema observado debido al acoplamiento, el proceso de simulación debe de ejecutarse a la máxima frecuencia y si esto implica que la frecuencia de renderizado sea superior a la necesaria entonces se limita la frecuencia de simulación potencialmente posible. Este paradigma continuo y acoplado consume todos los recursos de CPU disponibles en su afán de mostrar todos los cambios que suceden en el mundo virtual sin perder detalle.

En la Figura 1 se muestra un esquema básico de como funciona el bucle principal del núcleo de una AGTR monohílo bajo el viejo paradigma de ejecución.

Se puede deducir que existen algunas desventajas debido al paradigma continuo y acoplado:

- Desperdicio de Potencia de Calculo.
- QoS global en lugar de tener una QoS individual.

- Acoplamiento de las fases de Simulación y Visualización.
- Recorrido del grafo de escena de forma ineficiente.

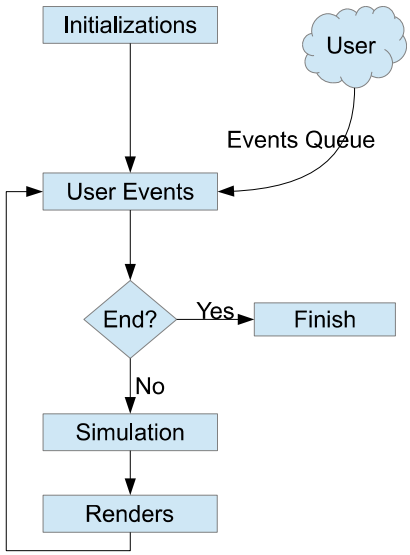

Figura 1: Bucle Principal Paradigma Continuo Acoplado

2.4.1 Desperdicio de la Potencia de Cálculo

El desperdicio de la potencia de cálculo genera un bajo aprovechamiento de la potencia de cálculo de la maquina. Toda la potencia de cálculo se utiliza para la fase de simulación y la fase de renderizado ejecutándolos a la máxima velocidad posible en lugar de discernir cuales son las verdaderas necesidades de los objetos de la aplicación. El reparto inadecuado de la potencia de cálculo puede provocar que los objetos no se simulen de una forma correcta, es decir, los objetos que necesiten una simulación más rápida se pueden submuestrear y los que necesiten una simulación más lenta se pueden sobremuestrear. Los objetos que se submuestrean pueden tener un comportamiento extraño en su visualización y los que se sobremuestrean desperdician potencia de cálculo que podría ser aprovechada por los submuestreados [13].

2.4.2 QoS Global vs QoS Individual

La imposibilidad de definir la QoS para cada aspecto de cada objeto y garantizar su requerimientos durante el tiempo de ejecución es otro de los problemas que tiene un

sistema continuo. El paradigma continuo acoplado trata a todos los objetos por igual y asume que las QoS de todos ellos son la misma, se trabaja con una QoS Global. Si el comportamiento de los objetos no cumple con el teorema de Nyquist Shannon, no se simularan adecuadamente produciendo una perdida de eventos no detectados.

Si se pudiera definir una QoS individual para cada objeto del sistema se podría conseguir una optimización del uso de la potencia de cálculo ya que cada objeto solo utilizaría los recursos de CPU necesarios según sus necesidades definidas previamente por el programador.

2.4.3 Problema del Acoplamiento

El problema del acoplamiento consiste en el acoplamiento de los procesos de simulación de todos los objetos del sistema incluyendo el de renderizado. El paradigma continuo acoplado produce un acople de las fases de simulación y visualización. La frecuencia de muestreo del sistema es la frecuencia de renderizado del sistema. No permite definir la frecuencia de renderizado ni garantiza que se mantenga durante la ejecución. Si bien es cierto que la mayoría de AGTR acoplan las fases de simulación y visualización, existen ciertas aplicaciones que, manteniendo un esquema continuo de simulación, permiten desacoplar estas fases. Pero el resto de la aplicación sigue manteniendo un esquema continuo. Este desacople se utiliza mucho en sistemas de realidad virtual con el objetivo de distribuir los procesos del sistema [13]. Algunas aplicaciones de realidad virtual que utilizan un modelo desacoplado son: MR ToolKit [14], Alice&Diver [15] o Bridge [16].

2.4.4 Problema en los Grafos de Escena

En las AGTR basadas en Grafos de Escena, la simulación se produce recorriendo el grafo de escena por lo que los objetos de las AGTR tienen prioridades en función de su situación caprichosa dentro del grafo de escena. Esta prioridad es ficticia e intrínseca al propio modelo de simulación. Los eventos se ejecutan en el orden en que se accede a los objetos en el grafo de escena. No están ordenados por tiempo de ocurrencia. Se recorren todos los objetos, independientemente de que estén activos o no, o tengan o no eventos pendientes. Esto supone ralentizar el proceso de simulación comprobando

objetos innecesariamente. Es muy ineficiente tener que recorrer todo el grafo de escena cuando muchos de los objetos no generarán nunca ningún evento. Algunas posibilidades de simulaciones erróneas son la ejecución desordenada de eventos o la ejecución de eventos cancelados [13].

2.5 Propuesta de Mejora

La serie de desventajas vistas en el apartado anterior llevan a plantear una posible hipótesis:

- Para optimizar la utilización de recursos de las AGTR, ¿seria posible y eficaz transferir la tecnología utilizada por la Simulación Discreta a el núcleo de gestión de las AGTR?

Esta transferencia tecnológica supone pasar de un paradigma de simulación continuo acoplado a un paradigma de simulación discreto desacoplado. Con este nuevo paradigma se consigue una optimización del uso de la CPU debido a que todos los elementos que forman parte de la AGTR son desacoplados y se ejecutan solo cuando es necesario de forma independiente [36].

Con el cambio de paradigma, los eventos suceden en el instante en que debe suceder. Se ejecutan ordenados en el tiempo. Se dispone de un gestor de eventos, de forma que sólo se atienden aquellos objetos que generan eventos, evitando el resto de objetos. Los objetos se atienden según el momento de ocurrencia del evento. Todos los objetos tienen la misma prioridad, incluyendo el usuario. No se pierden eventos, porque no hay submuestreo. En este caso puede suceder que el sistema sea tan complejo que se ralentice para poder simular el sistema adecuadamente [11].

No existe una QoS o frecuencia de muestreo fija y común a todos los elementos del sistema. Cada objeto tiene su propia frecuencia de muestreo (determinada por el programador). Los objetos con comportamientos rápidos se muestrean a la máxima frecuencia. Los objetos lentos se muestrean sólo cuando sea necesario, no sobrecargando el sistema con muestreos innecesarios. La frecuencia de muestreo es menos dependiente de la potencia de cálculo [11].

- Si la potencia de cálculo es insuficiente, el sistema funciona más lento pero no cambia la proporción de muestreo de los objetos. La simulación sigue siendo correcta.

- Si el sistema tiene mucha potencia de cálculo, en un esquema continuo son sobremuestreados y se desaprovecha la potencia de cálculo. En cambio, con un esquema de simulación discreta la simulación sólo consume la potencia de cálculo estrictamente necesaria.

- Independientemente de la potencia de cálculo, con un esquema de simulación discreta se simula sólo lo necesario. Los objetos no son submuestreados ni sobremuestreados.

Ademas, con el nuevo paradigma la AGTR puede adaptarse dinamicamente durante su ejecución. Se puede modificar la QoS o frecuencia de muestreo de cada objeto, ya sea un objeto gráfico o el objeto render o cualquier otro, en función de sus necesidades en cada instante de la ejecución, en función de su posición dentro del mundo virtual, en función de los recursos del sistema disponibles, …

El hecho de liberar recursos del uso de CPU genera una serie de ventajas para las AGTR como puedan ser mejorar la calidad, mejorar la portabilidad o hacer ahorro energético mediante técnicas de Green Computing (ver apartado 2.5.1):

- La calidad es mejorada debido a algunas posibilidades como:

    - Incrementar el numero de objetos.
    - Mejorar la IA a través de arboles de decisión más profundos
    - Mejorar la IA aumentando la frecuencia con la que ejecuta.
    - Incluir nuevos efectos gráficos usando materiales más sofisticados.
    - Utilizar para los objetos gráficos mallas más grandes.
    - Mejorar la frecuencia de simulación para cada objeto y asegurar que cada uno cumple con el Teorema del muestreo de Nequist Shannon de forma independiente.
    - …

- La portabilidad es posible ya que la AGTR puede aprovechar la optimización de

recursos para ejecutarse en plataformas más pequeñas como puedan ser dispositivos móviles o maquinas con unos años de antigüedad. Ademas, se puede prevenir una mejora en la calidad para su ejecución en maquinas nuevas con mejores prestaciones que cuando se desarrollo la aplicación.

- El ahorro energético es posible simplemente si no se hace uso de los recursos liberados. Ademas, en muchas de las maquinas actuales como servidores, ordenadores de sobremesa, ordenadores portátiles o dispositivos móviles se puede programar una reducción o ampliación de la utilización de recursos de la maquina en función de las necesidades del sistema (ver apartado 2.5.1).

Este nuevo paradigma sería completamente compatible con el estado actual de la tecnología dado que incide en cómo se gestionan los eventos internamente en el sistema. Es decir, que su aplicación es compatible con la tecnología actual y no requiere de ningún cambio tecnológico. Ademas, debido a los avances actuales para la Simulación Discreta en el campo de la computación paralela y distribuida [9], las AGTR implementadas con el nuevo paradigma pueden abrir nuevas lineas de I+D que posiblemente optimicen mucho más el rendimiento con CPUs multinúcleo y multihilo o en sistemas distribuidos.

2.5.1 Green Computing.

Green Computing también conocido como Green IT o traducido al español como Tecnologías Verdes se refiere al uso eficiente de los recursos computacionales minimizando el impacto ambiental, maximizando su viabilidad económica y asegurando deberes sociales. No sólo identifica a las principales tecnologías consumidoras de energía y productores de desperdicios ambientales sino que ofrece el desarrollo de productos informáticos ecológicos y promueve el reciclaje computacional. Algunas de las tecnologías clasificadas como verdes debido a que contribuyen a la reducción en el consumo de energía o emisión de dióxido de carbono son computación en nube, computación grid, virtualización en centros de datos y teletrabajo. En [17] hay una recopilación de todas las conferencias realizadas respecto a este tema.

Una de las características principales del nuevo paradigma propuesto es la

posibilidad de liberar trabajo de la CPU (en un sistema monohilo) o de los Cores (en un sistema multicore). Gracias a la librería ACPI (Advanced Configuration and Power Interface) se podría realizar un control sobre el consumo de energía de las maquinas donde se ejecutan las AGTR implementadas con el método discreto desacoplado.

ACPI es un estándar resultado de la actualización de APM a nivel de hardware, que controla el funcionamiento del BIOS y proporciona mecanismos avanzados para la gestión y ahorro de la energía. Va más allá de las posibilidades de APM. Así, por ejemplo, convierte la pulsación del botón de apagado en un simple evento, de tal forma que el sistema operativo puede detectarlo y le permite efectuar un apagado ordenado de la máquina, sin riesgo para el hardware de ésta como ocurría anteriormente [19].

Es una especificación industrial abierta desarrollada por las empresas HP, Microsoft, Intel, Phoenix y Toshiba. La ultima revisión es la 5.0 de diciembre de 2011 [19].

ACPI establece los estándares para la gestión de energía y disipación de calor de plataformas móviles, de sobremesa y servidores. Define modos de comportamiento a nivel Global, a nivel de dispositivo y a nivel de procesador.

A nivel de procesador define las siguientes opciones de configuración:

- Power states C0-C3
  - C0: estado de ejecución de instrucciones.
  - C1-C3: no se ejecutan instrucciones y a mayor número menor consumo, pero mayor coste de recuperación.
    - C1: Halt, recuperación inmediata.
    - C2: Stop-Clock, mantiene los estados del sotware visibles, recuperación lenta.
    - C3: Sleep, mantiene el orden de ejecución pero no la cache, recuperación muy lenta.
- Performance states Pn (0 <= n <= 16)
  - P0: Máxima potencia y frecuencia.
  - P1: menor que P0, voltaje y frecuencia escaladas.
  - P2: menor que P1, voltaje y frecuencia escaladas.

- …
- Pn: menor que Pn-1, voltaje y frecuencia escaladas.

Esta claro que si se habla de servidores e incluso ordenadores de sobremesa que normalmente están encendidos 24 horas el ahorro de energía es importante pero es fácil pensar que el ahorro de energía en un ordenador portátil o dispositivo móvil no supone demasiado ahorro, pero si se considera el número de estos equipos que pueden estar funcionando a la vez por todo el mundo, rápidamente se hace obvio el gran ahorro que puede suponer la optimización del uso de las CPUs.

3 RT-DESK

3.1 ¿Que es RT-DESK?

RT-DESK es el acrónimo de Real Time Discrete Event Simulation Kernel. Es el resultado de la transferencia tecnológica de la Simulación Discreta a la Simulación en Tiempo Real, es la herramienta desarrollada como soporte para poder hacer nuevas AGTR basadas en el nuevo paradigma discreto desacoplado o transformar AGTR ya implementadas en el nuevo paradigma. Básicamente es un núcleo de gestión de eventos ordenados temporalmente. Se presenta en forma de biblioteca de programación para ser integrada dentro de una aplicación. No es una aplicación en sí misma. Las aplicaciones que pueden hacer uso de esta tecnología son las ya citadas AGTR: típicamente videojuegos, realidad virtual o aumentada, aplicaciones científicas de simulación, entrenadores digitales, toma de decisiones a partir de simulaciones de comportamiento de sistemas,… Puede ser empleada tanto en entornos en tiempo real como para simulación científica de altas prestaciones no interactiva.

RT-DESK es un núcleo de simulación de aplicaciones gráficas en tiempo real, que, una vez integrado en una aplicación gráfica, gestiona los eventos del sistema. RT-DESK no puede funcionar de forma aislada, no tiene sentido si no se integra dentro de una aplicación. RT-DESK controla la comunicación de los objetos. Esta comunicación se realiza mediante paso de mensajes.

3.1.1 Funciones principales

Las funciones principales de RT-DESK son:

- Proporcionar a la AGTR las estructuras de datos y las funciones necesarias para modelar el mecanismo de pasos de mensajes.

- Gestionar la comunicación de los objetos mediante paso de mensajes:
  - Gestionar el proceso de envío de mensajes.
  - Mantener los mensajes enviados y todavía no recibidos por el objeto destino

ordenados por tiempo de ocurrencia.

- Enviar los mensajes a los objetos destino en el instante de tiempo indicado.
- Garantizar que los mensajes son enviados a los objetos receptores en el tiempo indicado y ordenados por tiempo.

RT-DESK se limita a gestionar los mensajes que se envían y reciben en el sistema. No realiza ninguna tarea que los mensajes lleven asociada. El objeto receptor sera quien ejecute la tarea que el propio objeto tenga asociada a ese mensaje. El objeto, en función del tipo de mensaje, procedente de un objeto emisor determinado que incluso puede ser el mismo, y con unos parametros determinados, seleccionara el proceso de respuesta al mensaje. Todo este proceso pertenece a la AGTR donde se integre RT-DESK y no al propio RT-DESK.

RT-DESK se utiliza para gestionar eventos de una AGTR pero tambien puede utilizarse en otro tipo de aplicaciones que necesiten paso de mensajes entre los distintos elementos de la aplicación. Si los mensajes llevan asociado un tiempo entonces los mensajes se enviaran en tiempo real y si llevan asociado timepo 0 entonces los mensajes se enviaran de forma inmediata.

3.1.2 Objetivos

Los objetivos principales de RT-DESK son:

- Gestionar los eventos de manera discreta.

- Ser autocontenido, de manera que sea facilmente integrable en otras AGTR. Debido a que debe poder adaptarse a diferentes tipos de AGTR parte de su funcionalidad debe poder delegarse en la AGTR.

- Permitir una gestión de eventos transparente al usuario. El programador debe interferir lo menos posible en el funcionamiento del núcleo.

3.1.3 Eventos del Sistema

El Evento de Sistema es un concepto que en RT-DESK corresponde a la definición del evento de la simulación de sistemas: es el cambio de estado de una entidad del sistema, una ocurrencia instantánea que altera el estado del sistema [21].

Uno de los objetivos de RT-DESK es la gestión de eventos de manera discreta. Para poder entender mejor la diferencia entre un evento discreto y un evento continuo a continuación se comenta un ejemplo: Se supone una luz que debe de encenderse pasados 20 segundos de un instante inicial. Si el evento es continuo, el sistema continuo debe de muestrear a la máxima velocidad posible si ya han transcurrido los 20 segundo mientras que si el evento es discreto, el sistema discreto programara un evento para dentro de 20 segundos y durante este tiempo no tendrá que realizar ninguna operación. De esta forma se consigue liberar a la CPU de un trabajo innecesario.

Con una AGTR implementada con RT-DESK todos los objetos del sistema se comportan como eventos discretos lo cual permite simular eventos continuos según una frecuencia de tiempo constante o permite ir modificando los tiempos de ejecución de eventos en función de las necesidades del sistema. En la aplicación pueden coexistir comportamientos de todo tipo bien en diferentes objetos como en el mismo objeto.

3.1.4 Características para su transferencia tecnológica

RT-DESK presenta una API en forma de una colección de ficheros cabecera que se pueden incorporar a cualquier aplicación AGTR o de simulación en general que se haya programado en lenguaje C++ o C#. C++ es el lenguaje es utilizado por una gran parte de las empresas que desarrollan Videojuegos o aplicaciones de Realidad Virtual. Junto con las cabeceras, se suministra además un fichero en formato DLL, por lo que el código fuente no puede ser accedido por parte de los programadores externos, aunque si pueden acceder a su funcionalidad.

3.2 Evolución de RT-DESK.

A partir del simulador de eventos discretos DESK [10][13], desarrollado en C++, orientado a simular sistemas mediante teoría de colas, se realizaron una serie de ajustes para incluir el tiempo real y poder soportar simuladores de entornos virtuales

interactivos al que se llamo GDESK [11][13]. Los cambios realizados fueron, ademas de la nomenclatura:

- La Gestión de Tiempos: En un simulador de eventos discretos el tiempo del simulador viene determinado únicamente por el tiempo de ocurrencia de los eventos. El reloj de simulación avanza cada vez que se produce un evento en el sistema. Los eventos se ejecutan consecutivamente, sin esperas entre ellos. Este tiempo no tiene relación alguna con el tiempo real, pues lo que interesa es el comportamiento del sistema para extraer conclusiones. En una AGTR, los eventos deben ocurrir en el instante de tiempo real en el que se ha indicado que ocurran. Por ello, el reloj de GDESK debe avanzar con la ocurrencia de eventos, al igual que en DESK, pero únicamente si el tiempo del sistema ha alcanzado el tiempo del evento.

- La definición de Objetos: La AGTR pasa a ser un conjunto de objetos que intercambian mensajes que activan eventos. Todos los objetos de la aplicación heredan de la clase base de GDESK permitiendo que todos los objetos hereden la gestión de mensajes entrantes y salientes, y ademas, todos los objetos se traten de una forma uniforme. Cuando un objeto recibe un mensaje este realiza una serie de tareas definidas por el programador en función del tipo de mensaje.

- La definición de Mensajes: Un mensaje modela la comunicación entre dos objetos o el comportamiento de un objeto. Un mensaje siempre lo genera un objeto y lo recibe un objeto. Parte de los atributos del mensaje son propios del simulador y parte los define el programador de la aplicación. Cuando un mensaje llega a un objeto este deja de tener utilidad y pasa a la pool de mensajes.

- Determinar la finalización de la ejecución: La finalización en GDESK debe de realizarse por medio de un evento de petición explicita del usuario o bien por una condición de error.

Ahora, a este simulador de tempo real se le han añadido una serie de mejoras para el mantenimiento y control de la simulación. Ha este nuevo avance se le llama RT-DESK.

Las mejoras introducidas sobre GDESK son:

- Temporizador interno de alta resolución para control y monitorización
- Preparación para añadir un Monitor del núcleo.
- Refactoring interno que ha cambiado la nomenclatura del API, optimizando algunas funciones, añadiéndose nuevas y generando documentación automática, tanto interna como externa que anteriormente no existía.

3.3 Funcionamiento de RT-DESK

RT-DESK utiliza dos entidades básicas para modelar el mecanismo de paso de eventos: objetos y eventos. Los eventos son mensajes enviados de un objeto a otro por lo que RT-DESK controla los objetos de la AGTR por paso de mensajes. RT-DESK mantiene los eventos ordenados en función del tiempo en el que han de ser ejecutados hasta que el tiempo transcurrido excede y entonces son enviados al objeto destinatario encargado de la ejecución. La estructura básica de RT-DESK es el Dispatcher. Es el encargado de realizar el paso de mensajes en el instante preciso marcado por un reloj de tiempo real y también es el responsable de mantener el orden temporal de todos los eventos pendientes de envío [11][13]. Cualquier objeto puede enviar mensajes de eventos a otros objetos o a si mismo.

Todos los elementos de la AGTR pasan a ser objetos independientes. Cada elemento del mundo virtual, el proceso de renderizado, la gestión de eventos, … ahora pasan a ser objetos independientes con su propia QoS definida. Al tener una QoS independiente los objetos solo consumirán el tiempo de CPU necesario para satisfacer sus propios requerimientos. En la Figura 2 se muestra un esquema de como funciona RT-DESK.

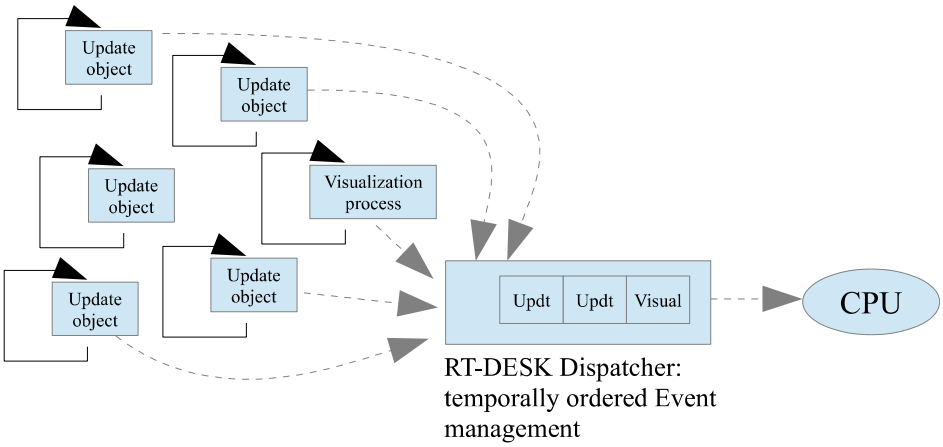

Figura 2: Esquema Funcionamiento RT-DESK

En este esquema se pretende mostrar como los objetos de la escena en el mundo virtual ejecutan su propia fase de simulación (Update object) enviándose mensajes a si mismos con una frecuencia de envío definida por su QoS para así poder cumplir sus propios requerimientos, también se muestra como la fase de renderizado (Visualization process) se comporta como un objeto más del sistema enviándose mensajes a si mismo con la frecuencia de visualización deseada. Todos ellos cuando necesitan ser ejecutados se mandan un mensaje a si mismo a través del Dispatcher de RT-DESK. Este ordena todas las peticiones en función del tiempo en el que deben de ser ejecutadas y cuando llega la hora devuelve un mensaje al objeto correspondiente para indicarle que ejecute. Cuando termina de enviar todos los eventos necesarios finaliza su ejecución y devuelve el tiempo que ha de transcurrir hasta su próxima llamada. De esta forma se garantiza que solo se ejecutan los procesos cuando son estrictamente necesarios en el avance de la ejecución.

Para entender un poco mejor cual seria la estructura de datos utilizada por RT-DESK ver el esquema de la Figura 3

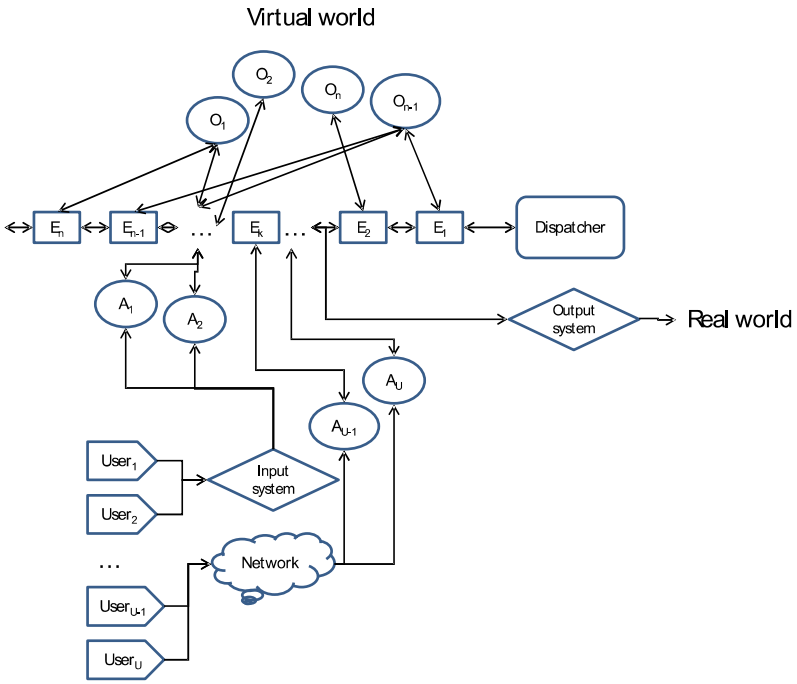

Figura 3: Esquema Estructura de Datos RT-DESK

En este esquema se pueden ver los objetos ($O_1..O_n$) que envían y reciben eventos ($E_1..E_n$) que son gestionados por el Dispatcher. Los posibles avatares ($A_1..A_n$) que representan a los usuarios conectados directamente o por red ($User_1..User_n$) también se comportarían como los objetos.

3.4 Tecnologías alternativas ya existentes en el mercado

Existen paquetes de simulación basados en teoría de colas, redes de Petri,… De entre ellos, se destaca:

- Flexim: Es una aplicación que engloba también herramientas de visualización de procesos, estadística de rendimientos, resultados y análisis para ayuda en la toma de decisiones. Es un programa líder en el mercado.

- GoldSim: Es un paquete de simulación generalista que dispone de extensiones para áreas específicas como la financiera, análisis de confianza, modelado de transportes masivos, flujo de contaminantes, así como herramientas complementarias para permitir la simulación distribuida o módulo de autor. Su funcionamiento interno, a raíz de la

funcionalidad que soporta, posiblemente sea parecido a la forma en la que trabaja el RT-DESK. Herramienta de apoyo a la toma de decisiones. No factible su integración en AGTRs.

- ProModel: Es un paquete de simulación generalista. Soporta simulación discreta y continua a la vez. Posiblemente, su funcionamiento sea parecido a RT-DESK. Herramienta de apoyo a la toma de decisiones. No factible su integración en AGTRs.

- Arena: Es una suite del holding Rockwell con capacidades de simulación de procesos industriales, transporte de mercancías,… Herramienta de apoyo a la toma de decisiones. No factible su integración en AGTRs.

- Extend: Equivalente a Arena.

- Witness: Aplicación de la casa Lanner. Su funcionamiento es parecido al resto de aplicaciones.

Se han hecho comparativas del funcionamiento interno de las aplicaciones de simulación indicadas anteriormente [34] [35]. Se puede afirmar que el esquema de funcionamiento es bastante parecido a la forma en la que trabaja RT-DESK, lo cual confirma que RT-DESK es una herramienta que está vigente hoy día y además, puede ser muy factible su transferencia tecnológica al mundo empresarial.

4 Benchmark

4.1 Objetivo

Este es un proyecto con orientación profesional, su objetivo es poder hacer una transferencia de la tecnología RT-DESK a la empresa. Con esta idea se ha realizado una AGTR basada en RT-DESK con la finalidad de tomar mediciones y realizar una herramienta de demostración que se dejara on-line para poder descargarla y así poder apreciar la versatilidad y respuesta del sistema.

4.2 Entorno de Software y Hardware

El desarrollo de la aplicación y las mediciones realizadas han sido en el entorno descrito a continuación:

Hardware:

ASSUS Notebook K52JK/K52JB Series
Intel Core i5 CPU, M430, @ 2,27GHz 2,27GHz
3,86 Gb RAM utilizables
ATI Mobility Radeon HD 5145, 1 GB

Software:

SO Windows 7 Home Premium, SP1, 64 bits
Microsoft Visual Studio 2008, C++
Glut 3.7.6 – Glew 1.5.6 – OpenGL 6.1.7600.16385
RT-DESK, C++

4.3 La Aplicación

La aplicación desarrollada parte de una practica realizada para la asignatura Programación Gráfica (PG) perteneciente a este Máster. La aplicación original esta implementada con Glut y OpenGL, funciona de forma continua acoplada siguiendo el viejo paradigma comentado en puntos anteriores y consiste en una tetera que gira sobre si misma, una luz que gira sobre la tetera y un objeto rectangular con función de pantalla donde se refleja la sombra de la tetera proyectada por la luz. El desarrollo realizado consiste en adaptar esta aplicación inicial en una aplicación que funcione de

forma discreta desacoplada mediante la utilización de RT-DESK o de forma continua acoplada. Puede lanzar de 1 a 25 teteras para poder incrementar las cargas de render y simulación y así poder hacer mediciones y comparar los dos modos de funcionamiento con diferentes cargas. Las mediciones son realizadas con contadores de alta resolución que van registrando los ticks de CPU entre dos instantes seleccionados con la idea de afectar lo menos posible a las mediciones. Al final de la ejecución es cuando se procesan los ticks registrados y se realizan los cálculos de los tiempos para luego generar una salida de archivos con estos resultados. El tiempo de ejecución por defecto es de aproximadamente 1 minuto.

4.3.1 Desarrollo

Lo primeo que se ha tenido que hacer es crear clases para todos los objetos de la escena ya que para el uso de RT-DESK se necesitan clases que hereden de la entidad básica de RT-DESK (_CEntity). No todas las clases generadas necesitan heredar de _CEntity, solo los objetos que pueden sufrir cambios en la fase de simulación dentro del mundo virtual y el objeto render son los que necesitan heredar, pero para una posterior reutilización se decidió implementar todas las clases necesarias. De esta manera todos los elementos de la aplicación tienen sus propios métodos y atributos y la aplicación principal solo tiene que ir pidiendo a cada objeto lo que necesita.

Las clases resultantes han sido las siguientes:

- Axis: Clase que permite generar unos ejes en el origen de coordenadas.
- Camara: Clase para crear y manejar la cámara de la aplicación.
- CargaUpdate: Clase diseñada para generar una carga artificial.
- FPS: Clase para calcular los Frames Por Segundo. También tiene un método de dibujado para mostrar datos en el mundo virtual.
- Glwindow: Clase para generar y remodelar una ventana de Glut.
- HRTimer: Clase diseñada para la utilización de contadores de alta resolución con posibilidad de tomas parciales y calculo de tiempos.
- Luz: Clase para dibujar una luz y proyectar la sombra que genera esta luz sobre un objeto en el receptor.
- Receptor: Clase para crear el objeto de fondo sobre el que se reflejan las sombras.

También se ha creado una clase Grid para colocar un grid sobre este.

- Render: Clase para definir el objeto Render de la aplicación. Es el encargado de realizar el proceso de visualización del mundo virtual.
- Tetera: Clase que define los objetos tetera.
- TipoMaterial: Clase para poder definir y aplicar un tipo de material OpenGL.
- TipoTextura: Clase para poder seleccionar y cargar una textura:

Todas las clases que representan objetos que se dibujan en el mundo virtual tienen el método draw() y draw_GdeskMsg(). El primero es el método que se llama para el dibujado del objeto en modo continuo y el segundo es el utilizado en el modo discreto. También pasa lo mismo con los métodos Update() y Update_GdeskMsg(). El motivo de esta separación es que los métodos _GdeskMsg() terminan lanzando un mensaje de evento indicando el tiempo de cuando se han de volver a ejecutar.

Para que la comparativa sea lo más objetiva posible se ha decidido integrar RT-DESK en el bucle principal de Glut llamado GlutMainLoop(). Gracias a esto, este trabajo también puede servir como una guía practica en la adaptación de RT-DESK en las AGTR ya implementadas con Glut.

Hay que destacar que una aplicación desarrollada desde el principio con RT-DESK no necesita adaptarse a ningún bucle ya existente. Se implementaría un bucle donde se pude aprovechar de una forma más óptima el tiempo disponible entre ejecuciones del Dispatcher.

Para entender mejor en que consiste esta adaptación se muestran las siguientes Figuras:

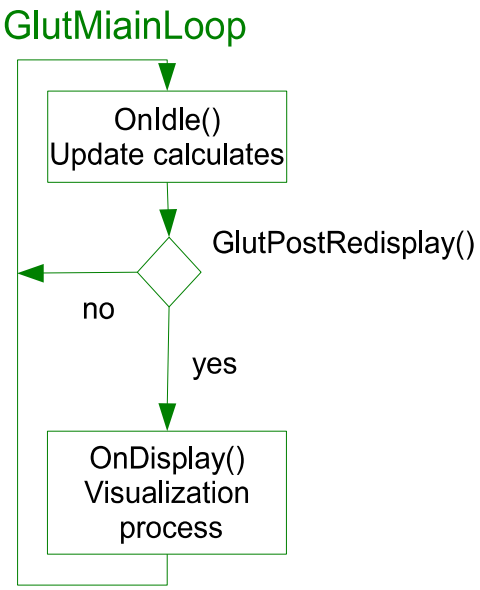

Figura 4: Bucle Principal Glut

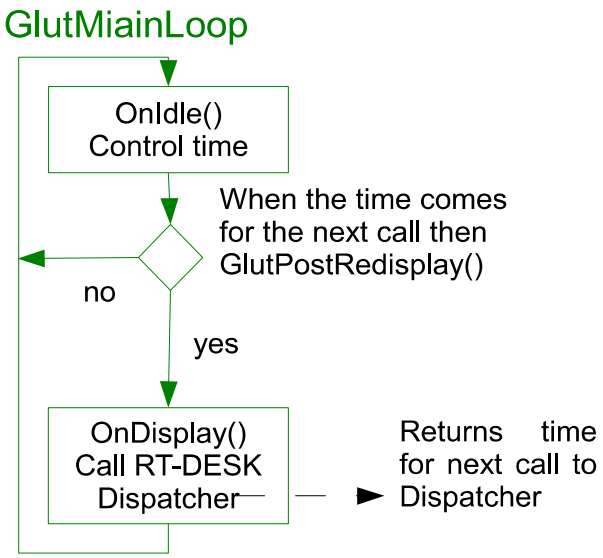

Figura 5: Bucle Principal Glut con RT-DESK Integtrado

La Figura 4 representa el bucle principal de Glut en su uso habitual (continuo desacoplado). Se puede observar como el sistema esta supervisando constantemente si se ha producido alguna actualización de algún elemento de la escena con OnIdle() (fase de simulación). En el momento que se realiza una modificación de cualquiera de los elementos de la escena se lanza el proceso de dibujado o visualización OnDisplay() (fase de renderizado). Si un elemento de la escena cambia muy rápido puede provocar una visualización demasiado rápida y por ello una sobrecarga de CPU que implica una menor velocidad en la supervisión. Aquí es donde se puede observar el problema del acoplamiento comentado en apartado 2.4.3 .

En la Figura 5 se adapta RT-DESK al bucle principal de Glut. El sistema esta supervisando constantemente con OnIdle() pero esta vez solo esta controlando el tiempo de ejecución (En este momento se pueden lanzar procesos de los que se conocen su tiempo de ejecución y que no sobrepasen el tiempo que queda para despertar al Dispatcher. Por ejemplo procesos de IA, permitir procesos de simulación más precisos, procesos de Green Computing, ...). Cuando llega el instante de tiempo indicado por el Dispatcher de RT-DESK se lanza una petición de dibujado que llama a OnDisplay(). Esta vez OnDisplay() lo único que hace es despertar al Dispatcher para que envíe los eventos pendientes a los objetos correspondientes. Cuando termina vuelve a indicar cuando cuando hay que despertarlo de nuevo. De esta forma cada objeto de la

aplicación ejecuta su simulación cuando lo necesita y el objeto render solo dibuja con la frecuencia definida. Aquí es donde se puede apreciar el desacoplo del que se habla en el apartado 2.5 .

4.3.2 Inicialización

Debido a que RT-DESK funciona con envío de mensajes de eventos, es necesario que cuando se inicialice la aplicación en modo discreto se lancen por primera vez todos los eventos necesarios para el correcto funcionamiento inicial. Una vez la ejecución esta en marcha es posible lanzar eventos nuevos o dejar de enviar eventos.

Las tareas básicas que se realizan en la inicialización son:

- Inicialización de Glut. En esta inicialización se dejan sin activar los eventos de teclado y ratón para no interferir por descuido en las mediciones que se realizaran luego.
- Asignación de posiciones en el mundo virtual a cada una de las teteras.
- Inicialización de los contadores de alta resolución que van a ser utilizados.

A partir de este punto las tareas son añadidos necesarios para el correcto funcionamiento de la aplicación con RT-DESK en modo continuo.

- Inicialización de RT-DESK. RT-DESK debe de ser inicializado para su correcto funcionamiento mediante la función GDESK_StartUpEngine(). Esta función se encarga de configurar RT-DESK, crear el gestor de memoria para envío de mensajes y crear el Distpacher de eventos.

```
int aux = 1;
    aux = GDESK_StartUpEngine();
    printf("GDESK_StartUpEngine devuelve %d\n", aux);
```

- Asignación de los tiempos con los que se van a iniciar los objetos luz y teteras (tiempos de update) y el objeto render (tiempo de render).

```
//**********************Definimos los tiempos de envio de mensajes GDESK
    //16.67->60hz, 40->25hz , 33,33->30hz
    for(i=0;i<DEF_NUM_TETERAS;i++){
        tete[i].timeGdeskMsg = DEF_MS_UPDATE_T;
    }
    luz.timeGdeskMsg = DEF_MS_UPDATE_L;
    render.timeGdeskMsg = DEF_MS_RENDER;
```

- Envío de los primeros mensajes de cada objeto.

```
//Envio de los primeros mensanjes de GDESK
    for(i=0;i<DEF_NUM_TETERAS;i++){
        tete[i].msg = tete[i].GDESK_GetMessageToFill();
        tete[i].GDESK_SendMessage(tete[i].msg,tete[i],tete[i].timeGdeskMsg);
    }
    luz.msg = luz.GDESK_GetMessageToFill();
    luz.GDESK_SendMessage(luz.msg,luz,luz.timeGdeskMsg);
    render.msg = render.GDESK_GetMessageToFill();
    render.GDESK_SendMessage(render.msg,render,render.timeGdeskMsg);
```

Para ver el código completo de toda la inicialización ver Anexo A – Código Inicialización.

4.3.3 Medición de Tiempos

Los tiempos que se van a medir son los tiempos de Render, Update e Idle que se definen como:

- Tiempo de Render, es el tiempo de ejecución empleado para realizar la fase de renderizado. Tiempo necesario para dibujar todos los elementos de una escena en el mundo virtual. La medición del tiempo de Render se realiza contando los ticks de CPU transcurridos desde que empieza el proceso de dibujado hasta que termina.

  En el caso del modo continuo se toma el tiempo en Ondisplay():

```
OnDisplay(){
        Toma de Tick 1()
        Codigo para dibujar los elementos de la escena
        Toma de Tick 2()
        Guardar toma parcial()
```

```
}
```

En el caso del modo discreto se toma el tiempo cuando el objeto Render lanza la ejecución de su método Draw_GdeskMsg():

```
Render::RecibeMsg{
    Toma de Tick 1()
    Draw_GdeskMsg()
    Toma de Tick 2()
    Guardar toma parcial()
}
```

- Tiempo de Update, es el tiempo de ejecución empleado para realizar la fase de simulación. Tiempo necesario para supervisar y calcular los cambios de los objetos no estáticos en el mundo virtual. La medición del tiempo de Update se realiza:

  En modo continuo contando los ticks de CPU desde que empieza el proceso de Update hasta el final del proceso.

```
OnIdle(){
    Control Frec. Simulación
        ProcesoUpdate(){
            Toma de Tick 1()
            Codigo para evaluación y calculo de la simulación
            Toma de Tick 2()
            Guardar toma parcial()
        }
}
```

  El control de Frecuencia esta al minimo, es decir, se permite que la CPU trabaje a toda la velocidad posible para realizar mediciones más objetivas.

En el caso del modo discreto se toma el tiempo de cada objeto cuando el objeto lanza la ejecución de su método update_GdeskMsg():

```
Objeto::RecibeMsg{
        Toma de Tick 1()
        update_GdeskMsg()
        Toma de Tick 2()
        Guardar toma parcial()
    }
```

En este modo se tendrán tantas tomas parciales como objetos gráficos de la escena.

- Tiempo de Idle, es el tiempo de ejecución no útil. Representa el porcentaje de CPU libre para otros posibles usos. La medición del tiempo de Idle se realiza:

  En modo continuo contando los ticks de CPU desde que empieza el proceso de Idle() hasta el final del proceso y restando el tiempo de update.

```
OnIdle(){
     Toma de Tick 1()
     Control Frec. Simulación
            ProcesoUpdate(){
                Codigo para evaluación y calculo de la simulación
            }
        Toma de Tick 2()
}
```

  En el modo discreto el tiempo de Update es el valor acumulado de todos los tiempos que devuelve el Dispatcher, es decir, todo el tiempo que el Dispatcher esta dormido.

Por último comentar que se ha calculado también un pequeño tiempo llamado tiempo no controlado que es el tiempo que Glut necesita para sus gestiones propias. Este tiempo se calcula restando al tiempo total los tiempos de Render, Update y Idle.

4.3.4 Archivos de Salida

Una vez finaliza la ejecución se procesan todas las mediciones parciales realizadas y se generan una serie de archivos con los resultados para su posterior análisis o tratamiento. Los archivos de salida pueden ser generados en formato de texto (.txt) o en formato de valores separados por comas (.csv).

Si se elige el formato de texto y el modo continuo se generan tres archivos que se explican a continuación:

- C_ResumenTiempos.txt, es el archivo principal donde se muestra un resumen de todos los tiempos analizados.
- C_TiemposRender.txt, registra todos los tiempos de render parciales que se han tomado durante la ejecución.
- C_TiemposUpdate.txt, registra todos los tiempos de update parciales que se han tomado durante la ejecución.

Si se elige el formato de texto y el modo discreto se generan cuatro archivos que se explican a continuación:

- D_ResumenTiempos.txt, es el archivo principal donde se muestra un resumen de todos los tiempos analizados.
- D_TiemposRender.txt, registra todos los tiempos de render parciales que se han tomado durante la ejecución.
- D_TiemposUpdLuz.txt, registra todos los tiempo de update parciales que ha necesitado el objeto Luz.
- D_TiemposUpdTet.txt, registra todos los tiempos de update parciales que ha necesitado los objetos tetera. En la siguiente imagen se muestra un ejemplo:

Si en lugar de elegir el formato de texto se elige el formato csv se genera en un archivo de salida una linea con los tiempos medidos. Si el modo es continuo el archivo se llama ResumenC.csv y si el modo seleccionado es discreto el archivo se llama ResumenD.csv. Se ha elegido este formato ya que es un estándar soportado por muchas herramientas para la creación de gráficas.

Para ver más información sobre los archivos csv generados ver el Anexo B – Salidas .csv

4.3.5 Definición de variaciones en la ejecución

Por último, antes de pasar al análisis de resultados, se explican las diferentes posibilidades que se ofrecen en el archivo Definiciones.h perteneciente a la aplicación. En este archivo se han programado diferentes definiciones de control y constantes que fijan el comportamiento de la aplicación en el momento de la compilación. A continuación se explican las definiciones de control:

- DEF_SALIDA_CSV, si esta definida los archivos de salida serán de tipo csv y si no lo esta serán de tipo texto.
- DEF_CONCARGA, si esta definida se activa una carga artificial en los procesos de update de todos los elementos gráficos del mundo virtual.
- DEF_MAX_RENDER, si esta definida la aplicación finaliza en función del numero de renderizados que se realizan.
- DEF_MAX_TIME, si esta definida la aplicación finaliza en función del numero de ticks de CPU consumidos, es decir, esta limitada por tiempo.

  Nota: Estas dos ultimas constantes no pueden estar definidas al mismo tiempo. Si no se define ninguna de ellas la aplicación no finalizara hasta que la pare el usuario.
- DEF_CONTINUO, si esta definida la aplicación se ejecutara en modo continuo acoplado.
- DEF_DISCRETO, si esta definida la aplicación se ejecutara en modo discreto desacoplado.

  Nota: Estas dos ultimas no pueden estar definidas a la vez y ademas debe de estar una de ellas definida.
- DEF_SOMBRAS, si esta definida la luz proyecta las sombras de las teteras sobre el receptor. Si no esta definida pues no las proyecta. Esta opción si se activa hay que tener cuidado con el numero de teteras utilizadas ya que puede sobrecargar el renderizado en exceso.
- GLFLUSH, si esta definida obliga a que se dibujen los elementos gráficos en la fase de render y no espera a que el hardware termine para continuar con el programa [12] de esta

forma se puede asegurar la máxima velocidad posible de ejecución en ambos modos.

Ahora se explican las constantes definidas:

- DEF_CARGA 20, el rango de valores probados son 1-20, entre estos valores se modifica la cantidad de carga virtual utilizada en caso de estar activada.
- DEF_LONG_ARRAYS 30000, longitud de los arrays utilizados para guardar medidas parciales. Con este valor el sistema es estable y garantiza hasta 25 teteras con un tiempo de ejecución no superior a un minuto.
- MAX_RENDER 1000, es el numero de renderizados que se realizan para finalizar la ejecución si esta activada esta opción.
- DEF_MAX_TICKS 120000000, es el numero máximo de ticks de CPU que se realizaran. Es decir, el tiempo máximo de ejecución si esta activada esta opción.
  Nota: 60000000->aprox 28seg 120000000->aprox 56seg
- DEF_MS_RENDER 40, es el valor en milisegundos de la frecuencia de envío de eventos del objeto render en modo discreto.
  Nota: 16.67ms->60hz, 40ms->25hz , 33,33ms->30hz
- DEF_MS_UPDATE_T 40, es el valor en milisegundos de la frecuencia de envío de los eventos de simulación de las teteras.
- DEF_MS_UPDATE_L 40, es el valor en milisegundos de la frecuencia de envío de los eventos de simulación de la luz.

5 Resultados

Para el análisis de resultados primero se han realizado las mediciones sin carga artificial y luego, para poder observar mejor algunos datos, con carga artificial. Para el modo discreto se han tomado mediciones a 25 y 60 fps. Para cada configuración se ha ejecutado la aplicación para 1, 3, 5, 10, 15, 20 y 25 teteras. En la siguiente tabla se puede ver un resumen de las cuatro configuraciones utilizadas.

	Nº teteras	FPS	Carga Artificial
Confg 1	1, 3, 5, 10, 15,20 ,25	25 -> 40 ms	no
Confg 2	1, 3, 5, 10, 15,20 ,25	25 -> 40 ms	SI
Confg 3	1, 3, 5, 10, 15,20 ,25	60 -> 16.6667 ms	no
Confg 4	1, 3, 5, 10, 15,20 ,25	60 -> 16.6667 ms	SI

Tambien comentar que para no tener problemas con el numero de teteras se han realizado todas las mediciones con la proyección de sombras desactivada.

5.1 Sin carga artificial

Los resultados en modo continuo indican que del 100% del tiempo total de ejecución, el 99% es tiempo de render, 1% es tiempo de update y 0% tiempo de idle (ver Figura 6). Debido a la falta de carga artificial todo el tiempo se emplea en dibujar la escena a la maxíma velocidad posible.

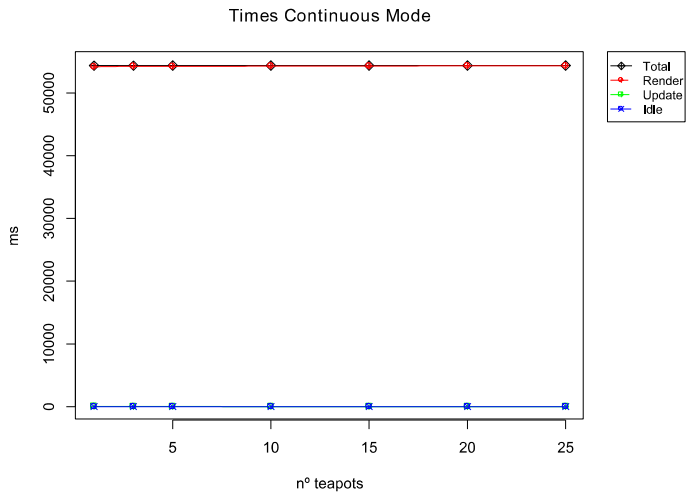

Figura 6: Tiempos Modo Continuo Sin Carga

En las siguientes figuras se puede observar claramente la diferencia de comportamiento entre el modo continuo y los modos discretos (Figuras 7 y 8).

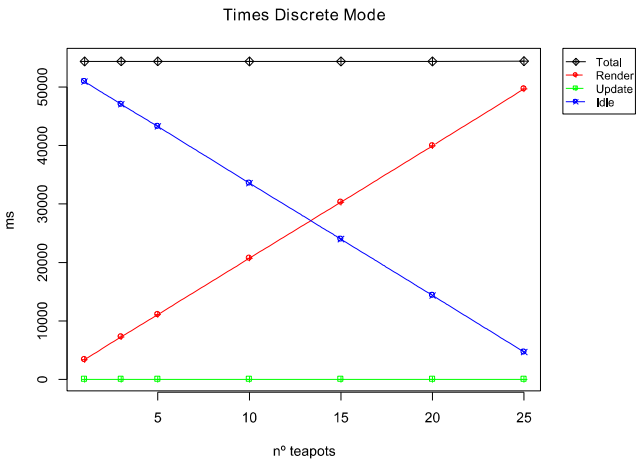

Figura 7: Tiempos Modo Discreto Sin Carga a 25 fps

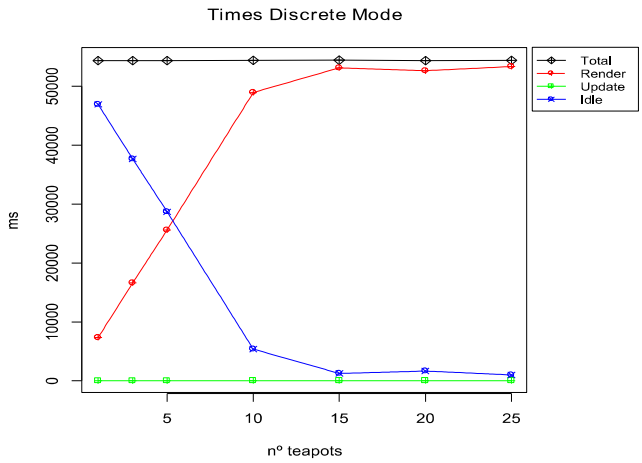

Figura 8: Tiempos Modo Discreto Sin Carga a 60 fps

La Figura 7 muestra los tiempos utilizados en modo discreto con una frecuencia de renderizado de 25 fps (Confg 1 de la tabla) y la Figura 8 es con una frecuencia de 60 fps (Confg 3 de la tabla).

Se puede apreciar como en el modo continuo (Figura 6) como prácticamente el tiempo total esta siendo utilizado para renderizado y update de manera que no queda tiempo libre (tiempo de Idle) disponible para nada. Debido a la poca carga de update generada no se aprecia bien el acoplamiento entre la fase de render y la de simulación. En el siguiente apartado se genera una carga artificial para poder apreciar mejor este acoplamiento.

En las Figuras 7 y 8 se puede observar como solo se utiliza el tiempo de render necesario para cumplir con los fps deseados generando de esta forma un tiempo libre en el uso de CPU. También se puede apreciar que ya no existe acoplamiento entre las fases de render y simulación. En el siguiente apartado se genera una carga artificial para poder apreciar mejor este desacoplamiento.

En la Figura 8 se aprecia que a partir de 10 teteras el sistema se satura y, aunque en tiempo de simulación se siguen realizando 60fps, en realidad, debido a la saturación del sistema informático, en tiempo real sólo pueden realizarse menos, generando una sensación de ralentización de la simulación. Esto se podría solucionar mediante una degradación suave de las exigencias de la Calidad de Servicio de la frecuencia de visualización, es decir, los fps.

Otras gráficas interesantes son las Figuras 9 y 10. En estas gráficas se puede observar como en modo continuo la frecuencia de renderizado siempre se ejecuta a la velocidad máxima permitida por el sistema mientras que en modo discreto se mantiene siempre en la frecuencia deseada Figura 9 a 25 fps y Figura 10 a 60 fps.

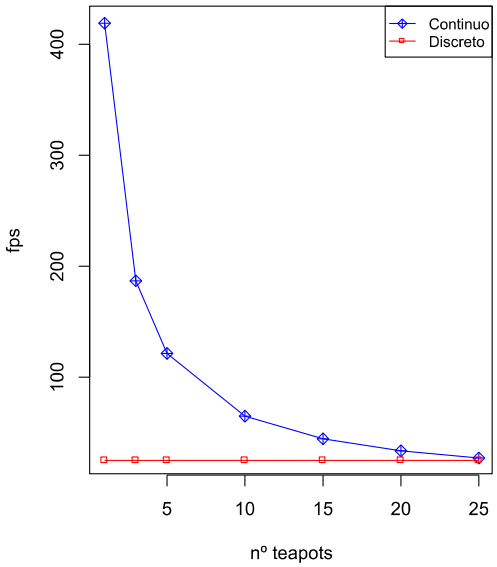

Figura 9: FPS Modo Continuo vs Modo Discreto a 25 fps

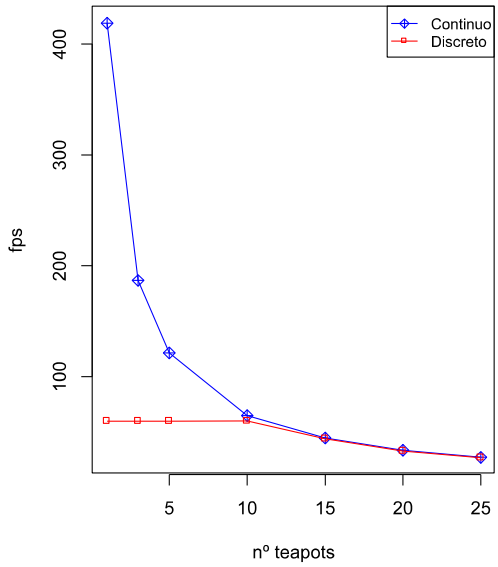

Figura 10: FPS Modo Continuo vs Modo Discrteo a 60 fps

En la Figura 10 se puede apreciar que cuando el sistema esta demasiado sobrecargado y no puede mantener la frecuencia de render deseada este puede autorregularse y comportarse como lo haría en modo continuo.

5.2 Con carga artificial

Los siguientes resultados analizados se han obtenido aplicando una carga artificial de update. Gracias a esta carga se puede apreciar mejor el acoplamiento en

modo continuo (Figura 11). Se sigue observando como no queda tiempo libre.

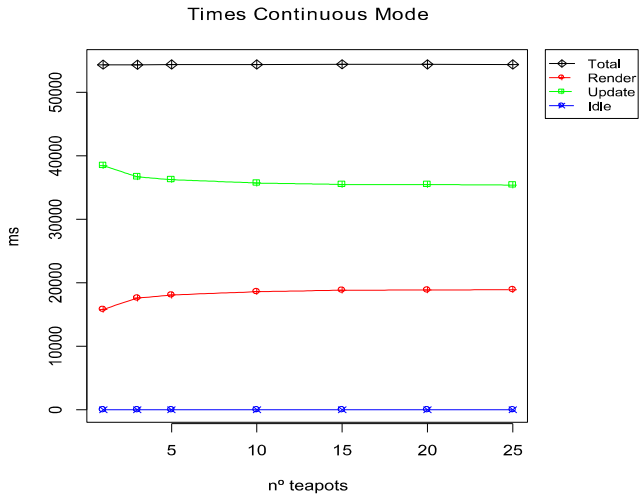

Figura 11: Tiempos Modo Continuo Con Carga Artificial

En las Figuras 12 a 25 fps y 13 a 60 fps se puede apreciar por un lado como están desacopladas las fases de renderizado y simulación y por otra parte como el sistema es capaz de autorregularse cunado se satura.

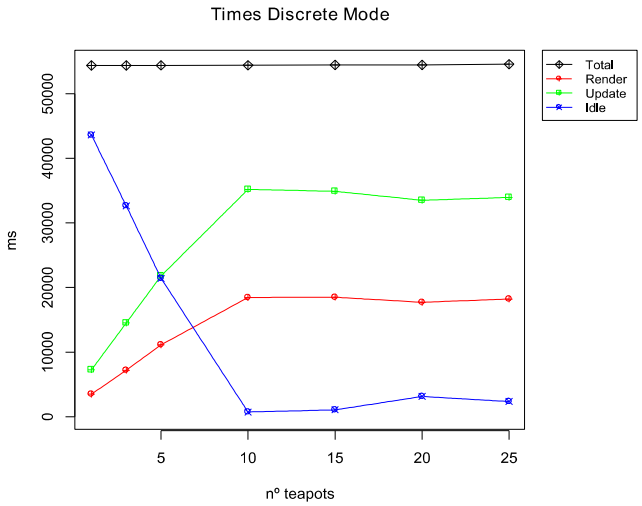

Figura 12: Tiempos Modo Discreto Con Carga a 25 fps

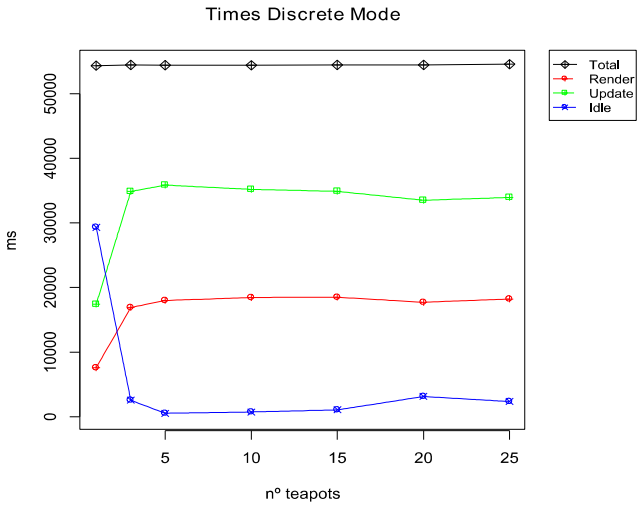

Figura 13: Tiempos Modo Discreto Con Carga a 60 fps

5.3 Comparativa con DFly3D

A continuación se presentan unas de las gráficas obtenidas en la Tesis de Inmaculada García [13] [36] que integraba GDESK en el motor de Videojuegos Fly3D y así poder comparar los resultados obtenidos en aquellos momentos con los obtenidos en esta Tesina que integra RT-DESK (que es la evolución de GDESK) sobre una AGTR basada en Glut/OpenGl. La integración de Fly3D con GDESK se le denomina Dfly3D.

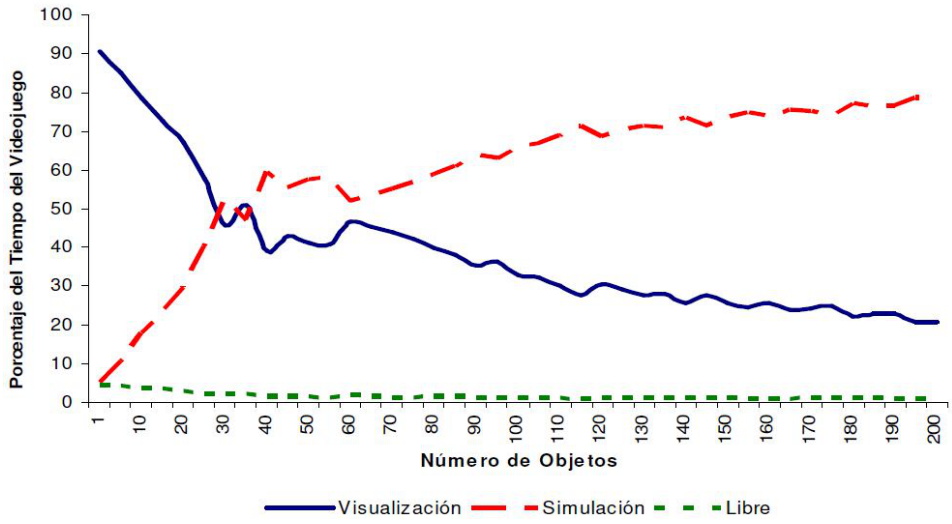

Figura 14: Resultados DFly3D en Modo Continuo

En la figura 14 se puede apreciar como existe el acoplamiento entre las fases de Simulación y Visualización cuando se trabaja en modo continuo. También se puede apreciar que no queda tiempo libre para la CPU independientemente del numero de objetos que participen.

Si comparamos la figura 11 que contienen los datos de el presente trabajo, con la figura 14 que contiene los datos obtenidos hace unos años y con diferentes tecnologías para el desarrollo de aplicaciones gráficas en tiempo real, podemos ver que el comportamiento es similar en ambas AGTR. Se confirma el problema del acoplamiento y el problema de utilizar excesiva potencia de cálculo. Un incremento en la carga de simulación supone un decremento del tiempo destinado a la visualización. Un incremento en la carga de visualización supone un decremento del tiempo de

simulación. Si el muestreo de cada objeto no es suficiente para cumplir el teorema de Niquist-Shannon, algún objeto puede estar muestreándose insuficientemente. Si la frecuencia de visualización no es suficiente, el sistema puede no visualizarse correctamente.

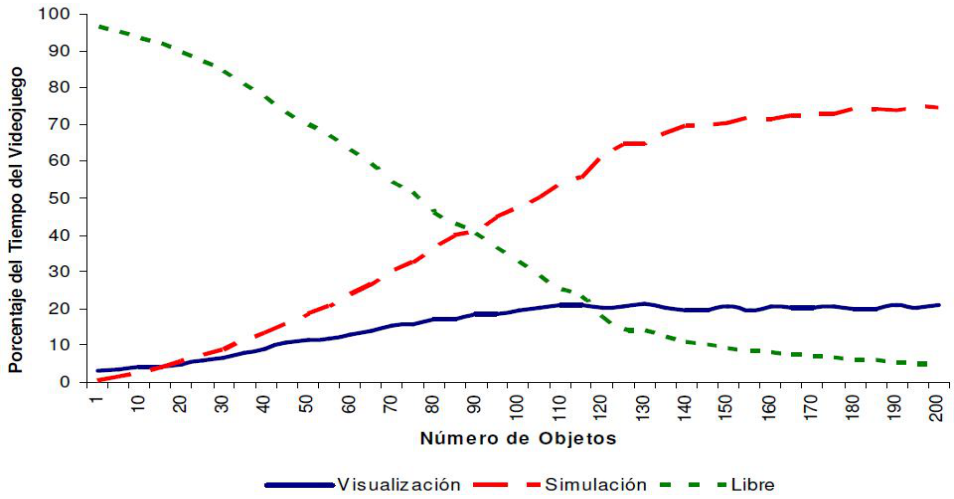

Figura 15: Resultados DFly3D en Modo Discreto

En la figura 15 se puede ver como utilizando el Modo Discreto se consigue solucionar el problema del acoplamiento y además, se libera tiempo de CPU optimizando así la potencia de cálculo. Conforme se incrementa el numero de objetos de la aplicación vemos como disminuye el tiempo libre de CPU lo cual es correcto.

Si comparamos la figura 12 con la figura 15 se puede observar que el comportamiento es el mismo. Ya no existe acoplamiento entre las fases de simulación y visualización y también se observa la optimización del tiempo de cálculo liberando uso de CPU cuando no es necesario. Se confirma la solución del problema de acoplamiento y el uso optimizado de la potencia de cálculo así como también se validan los resultados obtenidos en [13].

6 Conclusiones

6.1 Problema planteado

Las aplicaciones gráficas en tiempo real tradicionales, como los videojuegos, Realidad Virtual, Simulación 3D, Realidad Aumentada, suelen seguir un paradigma de simulación continuo y acoplado. En a mayoría de casos esta simulación continua genera un acoplamiento de las fases de visualización y simulación, aunque algunas aplicaciones permiten desacoplar estas fases.

Utilizar el paradigma de simulación continuo y acoplado ha mostrado claramente algunos problemas de ineficiencia debido fundamentalmente a las siguientes razones:

- En las AGTR que se acoplan las fases de simulación y visualización, la frecuencia de renderizado es igual a la frecuencia de muestreo del sistema. El programador no tiene control sobre la frecuencia de renderizado que garantice la QoS de la aplicación.

- La aplicación gráfica depende mucho de la potencia de cálculo de la máquina. El reparto de la potencia de cálculo entre los objetos de la aplicación no se realiza en función de las necesidades de cada objeto.

- El sistema puede presentar comportamientos incorrectos, como ejecución de eventos desordenados o eventos no detectados.

- Todos los objetos se muestrean a la misma frecuencia, independientemente de sus necesidades de QoS. Si la QoS de un objeto necesita una frecuencia de muestreo inferior a la frecuencia de muestreo del sistema, el objeto tendrá un comportamiento correcto, pero se generarán muestreos innecesarios. Si la QoS del objeto es superior a la frecuencia de muestreo, el objeto tendrá un comportamiento anómalo.

- Si la potencia de cálculo de la máquina es elevada para las necesidades de la aplicación gráfica se desperdicia potencia de cálculo. El sistema está continuamente simulando y visualizando a la máxima velocidad. Se generan escenas que nunca se visualizan y

simulaciones innecesarias.

- El sistema no es capaz de manejar las situaciones de colapso del sistema y adaptarse dinámicamente a estas, redefiniendo la QoS de cada objeto de forma independiente.

6.2 Solución propuesta

Este trabajo invita a cambiar el paradigma continuo y acoplado típico por el discreto desacoplado propuesto. La nueva aplicación tendrá un comportamiento discreto que permite el desacoplo de las fases de simulación y visualización lo cual permite liberar potencia de cálculo y permite mucha versatilidad en la definición de los comportamientos de los objetos y de todo el sistema en general.

6.3 Tecnología utilizada

RT-DESK es la tecnología que nos permite este cambio de paradigma, discretiza la AGTR donde es integrado. Permite aplicar a las AGTR las ventajas de la simulación discreta desacoplada para soportar en tiempo real aplicaciones interactivas como simuladores 3D, realidad virtual, mundos virtuales, realidad aumentada o Videojuegos.

La simulación discreta permite modelar comportamientos discretos, continuos e híbridos. Una consecuencia de la discretización del sistema es el desacoplo de las fases de visualización y simulación. RT-DESK gestiona los eventos del sistema haciendo que los objetos de la aplicación gráfica donde se integra se comuniquen mediante paso de mensajes. RT-DESK controla el proceso de envío y recepción de mensajes. Los mensajes tiene asociado un tiempo que indica el instante en que deben ser recibidos por el objeto receptor. RT-DESK captura los mensajes enviados y los almacena hasta que se cumple el tiempo del mensaje. En ese instante envía el mensaje al objeto receptor. RT-DESK sólo trata con mensajes, no realiza los procesos que el objeto receptor tenga asociados al mensaje.

El mecanismo de paso de mensajes modela tanto comportamientos continuos como comportamientos discretos de los objetos. Este mecanismo ofrece dos utilidades, por un lado ofrece la utilidad de comunicar objetos entre si y por otro lado permite modelar el comportamiento de un objeto.

La comunicación entre objetos permite, por ejemplo, que un objeto le diga a otro que cambie su comportamiento. Esto puede ser muy útil ya que pueden haber objetos pasivos que simplemente esperan que objetos activos les pidan actuar de una forma u otra en función de sus necesidades. Los objetos pasivos liberan potencia de cálculo ya que no necesitan un muestreo periódico.

El modelado del comportamiento de los objetos permite definir que un objeto se comporte de modo continuo o modo discreto o modo híbrido:

- Modo Continuo: Implica que el objeto debe actualizarse cada cierto tiempo. Para ello el objeto se envía un mensaje a si mismo para modificar su comportamiento cada T unidades de tiempo. Es decir, cuando recibe el mensaje ejecuta la tarea correspondiente y acto seguido vuelve a enviar otro mensaje.

- Modo Discreto: Implica que el objeto puede programar un comportamiento para que suceda pasadas T unidades de tiempo. Si un objeto desea programar un evento para un instante T, este solo tiene que enviar al objeto responsable un mensaje o también se lo puede enviar a si mismo. Como consecuencia del mensaje el objeto destinatario ejecutara la tarea definida en el mensaje y una vez ejecutada no volverá a enviar otro mensaje. El no volver a enviar mensaje lo diferencia del modo continuo.

- Modo Híbrido: Es una combinación de los dos anteriores. De hecho, en una AGTR compleja donde participan muchos objetos es el modo más óptimo ya que si se hace un buen diseño habrán objetos pasivos y objetos activos.

RT-DESK proporciona la posibilidad a la AGTR que lo integre de poder monitorizar tanto off-line como on-line. Con la monitorización off-line se pueden generar unos históricos con los resultados de la monitorización del sistema. Esto es muy útil en el proceso de implementación. Con la monitorización on-line se pueden definir mecanismos que obtengan información del sistema y así poder ajustar dinamicamente el comportamiento del sistema.

6.4 Análisis de resultados

Una vez analizados los resultados del Benchmark y los resultados obtenidos en la Tesis de Inmaculada García [13] se puede concluir que el cambio del paradigma continuo acoplado al nuevo paradigma discreto desacoplado puede ser muy útil y eficaz en el desarrollo de las AGTR. Los datos analizados demuestran que se puede optimizar mucho la ejecución de la fase de renderizado, también se puede optimizar las fases de simulación de los diferentes objetos, así como la simulación de los diferentes aspectos de cada uno de los objetos pertenecientes a una AGTR de una forma totalmente independiente permitiendo optimizar la potencia de cálculo liberando mucho tiempo de CPU que se puede emplear en otros menesteres.

Gracias al análisis realizado se confirma los resultados y la validez de la tecnología utilizada, tanto en este trabajo como en la Tesis de Inmaculada García [13]. Esta validación se realiza a pesar del tiempo pasado, del cambio tecnológico tanto en hardware como en software, de ser aplicaciones realizadas de forma completamente independiente y de haber empleado tecnologías para el desarrollo de aplicaciones gráficas en tiempo real diferentes.

6.5 Experiencia personal

Una vez se comprende y asimila esta nueva forma de plantear la programación, la implementación de RT-DESK es muy sencilla y da mucha versatilidad al programador para configurar cualquier aspecto de la aplicación.

7 Trabajos Futuros

El trabajo realizado abre las puertas de trabajos necesarios y de trabajos que pueden añadir ventajas más interesantes.

Como trabajos necesarios surge la necesidad de:

- Probar esta tecnología en AGTR más complejas y así poder añadir un monitor que controle el núcleo de RT-DESK para evitar posibles sobrecargas. Después de haber programado con RT-DESK y de haber efectuado multitud de pruebas de funcionamiento se observa la necesidad de desarrollar un monitor para el núcleo de RT-DESK. Este Monitor deberá de controlar que ante un exceso de eventos el núcleo no se quede en un bucle infinito sirviendo mensajes de eventos. Si llega a ocurrir, el resultado es que una vez se llama al Dispatcher (ver apartado 3.3) este entra en un bucle infinito ya que en su cola de eventos temporales siempre habrá un evento cuyo tiempo de ejecución sera inferior al tiempo real.

- Realizar un estudio de como poder integrar esta tecnología dentro de AGTR basadas en un grafo de escena. Debido a la gran cantidad de AGTR que actualmente utilizan los grafos de escena se considera que es muy importante para la evolución de esta tecnología y su posterior transferencia tecnológica el hacer un estudio detallado de las ventajas e inconvenientes que esto pueda suponer.

- Seria importante poder incorporar un sistema de seguridad con Watch Dog antibloqueo. Debido a la cantidad de aspectos que un Monitor del núcleo puede tratar, este desarrollo puede tardar más tiempo de lo deseado en su implementación y puesta apunto. Para que se pueda mientras tanto poder ir utilizando esta tecnología en otros trabajos, se considera muy necesario la incorporación de un sistema de seguridad.

- Mejorar la funcionalidad y la apariencia de la Demo implementada para presentar mejor la información deseada. Mostrar en lugar de datos numéricos unas gráficas que se actualicen on-line seria una mejora muy interesante ya que incrementaría el entendimiento del usuario y tendría un impacto visual mucho mejor.

Como lineas de trabajo muy interesantes podrían plantearse:

- La adaptación a GPGPU (General Porpouse). Dados los avances de hardware en el desarrollo de tarjetas gráficas junto con las posibilidades que estas ofrecen y la tendencia actual de utilizar sus procesadores como soporte de cálculo, se considera que esta tecnología podría aprovechar ampliamente los recursos ofrecidos por estos procesadores y mejorar la Calidad de Servicio, en todos sus aspectos, de las AGTR.

- La adaptación a sistemas multicore tanto para el reparto de hilos de ejecución como para el control energético de los procesadores. Se considera que este es un aspecto fundamental para la evolución de esta tecnología. Aprovechar al máximo las posibilidades que ofrecen las CPU actuales se convierte en una obligación por parte de los desarrolladores de software. Para que esta tecnología tenga buenas expectativas de evolución es muy importante añadir los mecanismos necesarios para facilitar al programador la gestión del reparto de hilos de ejecución entre los diferentes cores que ofrezca el sistema donde se ejecute.

- Agregar al monitor del sistema el manejo de la ACPI (ver en apartado 2.5.1) para realizar un control energético sobre las maquinas donde son ejecutados los programas y generar aplicaciones que cumplan con la iniciativa Green Computing. Esto disminuye la cantidad de recargas de batería en dispositivos móviles como consecuencia de la disminución del consumo y por lo tanto un aumento de su vida útil. La disminución del consumo también puede incrementar el tiempo de juego en dispositivos móviles. Para ver mejor la importancia de adaptarse a la iniciativa Green Computing, a continuación se citan dos textos que provienen de un Informe de Vigilancia Tecnológica realizado por la Universidad Politécnica de Madrid.

  “Teniendo en cuenta la necesidad de energía a nivel mundial para mantener las tecnologías TI (Tecnologías de la Información) y debido al continuo incremento del precio de la energía, la iniciativa Green IT se considera un necesidad a la hora de desarrollar nuevas tecnologías a nivel global.” [37].

  “Hoy en día las tecnologías de la información no pueden permanecer ajenas a la enorme

problemática existente en el medio ambiente: contaminación, calentamiento global, efecto invernadero, etc. La iniciativa Green IT pretende contribuir de forma sustancial al cuidado y mantenimiento de los ecosistemas naturales desde los equipos de las tecnologías de la información, haciendo posible un desarrollo sostenible. Dentro de las políticas que se siguen en Green IT, una de las más importantes es la que promueve un aumento de la eficiencia energética de los equipos electrónicos, pues las fuentes de energía eléctrica son una de las principales causas del increíble aumento de la huella contaminante en el planeta.” [37].

8 Referencias

[1] Valente, L., Conci, A., and Feijó, B. 2005. Real time game loop models for single-player computer games. In Proceedings of the Fourth Brazilian Symposium on Computer Games and Digital Entertainment, 89–99

[2] “A game loop architecture for the GPU used as a math coprocessor in real-time applications”, Marcelo P. M. Zamith, Esteban W. G. Clua et Al. Computers in Entertainment (CIE) - SPECIAL ISSUE: Media Arts archive, Volume 6 Issue 3, October 2008, Article No. 42

[3] The gpu used as a math co-processor in real time applications. M. Zamith, E. Clua, P. Pagliosa, A. Conci, A. Montenegro, and L. Valente. Proceedings of the VI Brazilian Symposiumon Computer Games and Digital Entertainment, pages 37–43, 2007.

[4] Automatic Dynamic Task Distribution between CPU and GPU for Real-Time Systems. Joselli, M. et Al. 2008 11th IEEE International Conference on Computational Science and Engineering. Pág. 48 – 55

[5] http://emonet.biology.yale.edu/agentcell/

[6] BetaSIM: A framework for discrete event modeling and simulation. Wolfgang Kreutzer, Kasper Østerbye. Simulation Practice and Theory, Volume 6, Issue 6, 15 September 1998, Pages 573-599

[7] Agent-based simulation tutorial - simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. Proceedings of the 2010 Winter Simulation Conference. Pág. 135 - 150

[8] Tools of the Trade: A Survey of Various Agent Based Modeling Platforms. Cynthia Nikolai and Gregory Madey. Journal of Artificial Societies and Social Simulation vol. 12, no. 2 2. 2009

[9] Exploiting concurrency in the implementation of a discrete event simulator. I. Castilla, F.C. García, R.M. Aguilar. Simulation Modelling Practice and Theory 17 (2009) 850-870. Elsiver.

[10] D.E.S.K. Discrete Events Simulation Kernel. I. Garcia, R. Mollá, E. Ramos y M. Fernández. European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS. Barcelona 11-14 Septiembre 2000.

[11] GDESK Game Discrete Event Simulation Kernel. I. Garcia, R. Mollá y T. Barella. Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972, February 2-6, 2004, Plzen, Czech Republic.

[12] http://dbarrero.tripod.com/uasimanimtr/MiniIntroOpenGL.pdf

[13] Simulación Híbrida Como Nucleo de Simulación de Aplicaciones Gráficas en Tiempo Real. Tesis Doctoral de Inmaculada García García. Universidad de Valencia, Servicio de publicaciones, 2004

[14] Shaw, C., Liang, J., Green, M., Sun, Y., ”The Decoupled Simulation Model for Virtual Reality Systems”, CHI Proceedings 1992.

[15] Randy, P., Conway, M., DeLine, R., Gossweiler, R., Maile, S., Ashton, J., Stoakley, R., .Alice & DIVER: A Software Architecture for the Rapid Prototyping of Virtual Environments”, Course notes for SIGGRAPH’94 course, Programming Virtual Worlds, 1994.

[16] Darken, R., Tonnesen, C., Passarella, K., ”The Bridge Between Developers and Virtual Environments: a Robust Virtual Environment System Architecture”, Proceedings of SPIE 1995.

[17] http://www.greenit-conferences.org/

[19] http://www.acpi.info/

[21] G.S. Fishman. Conceptos y Métodos en la Simulación Digital de Eventos Discretos. Limusa, Mexico, 1978.

[22] C. Shaw, J. Liang, M. Green, Y. Sun. The decoupled simulation model for virtual reality systems. CHI, 1992.

[23] R. Darken, C. Tonnesen, K. Passarella. The bridge between developers and virtual environments: a robust virtual environment system architecture. SPIE, 1995.

[24] G.G. Robertson, S.K. Card, J.D. Mackinlay. The cognitive coprocessor architecture for interactive user interface. UIST, 1989.

[25] J.B. Lewis, L. Koved, D.T. Ling. Dialogue structures for virtual worlds. CHI, 1991.

[26] M. Agus, A. Giachetti, E. Gobbetti, G. Zanetti. A multiprocessor decoupled system for the simulation of temporal bone surgery. Computing and Visualization in Science, 5(1), 2002.

[27] L. von Bertallanfy. General System Theory. Foundations, Development, Applications. Braziller, New York, 1968.

[28] A.M. Law, W.D. Kelton. Simulation Modeling and Analysis. McGraw-Hill Series in Industrial Engineering and Management Science, 1982.

[29] A.A.B. Pritsker. The GASP IV Simulation Language. John Wiley and Sons, Inc., New York, 1974.

[30] J. Banks, J.S. Carson II, B.B. Nelson, D.M. Nicol. Discrete Event System Simulation. Prentice Hall International Series in Industrial and Systems Engineering, 2001.

[31] T.H. Naylor, J.L. Balintfy, D.S. Burdick, K. Chu. Técnicas de Simulación en computadoras. Limusa, Mexico, 1975.

[32] T.J. Schriber, D.T. Brunner. Inside discrete event simulation software: how it works and why it matters. Proceedings of the 1999 Winter Simulation Conference, 1999.

[33] G.S. Lee. Towards an integration of computer simulation with computer graphics. Proceedings of the Western Computer Graphics Symposium, 1999.

[34] Inside discrete-event simulation software: how it works and why it matters. Thomas J. Schriber y Daniel T. Brunner. Proceedings of the 1997 Winter Simulation Conference.

[35] Inside discrete-event simulation software: how it works and why it matters. Thomas J. Schriber y Daniel T. Brunner. Proceedings of the 2011 Winter Simulation Conference.

[36] I. García, R. Mollá. Simulación desacoplada de eventos discretos en videojuegos. In CEIG, 2004.

[37] J. Garbajosa, E. Huedo, M. López. Green IT: Tecnologías para la Eficiencia Energética en los Sistema TI. Universidad Politécnica de Madrid, 2008.

9 Anexo A – Código Inicialización

Código completo del método de inicialización del Benchmark.

```
void init(void)
{
    //**********************************Inicialización de OpenGL
    glClearColor (0.7, 0.7, 0.7, 0.0);
    if (textu.loadTexture(&textu.tex)) exit(-1);
    glShadeModel (GL_SMOOTH);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambient);
    luz.init();
    glCullFace(GL_BACK);
    glutDisplayFunc(onDisplay);
    glutReshapeFunc(onReshape);
    //glutKeyboardFunc(onKeyDown);
    //glutMouseFunc(onMouse);
    //glutMotionFunc(onMouseMove);
    ctrlFPS.initTime=glutGet(GLUT_ELAPSED_TIME);
    updateTime=glutGet(GLUT_ELAPSED_TIME);
    glutIdleFunc(onIdle);

    //*******************************Asignamos posición a las teteras*******************
    for(i=0;i<DEF_NUM_TETERAS;i++){
        tete[i].posicion(PosTete[i].x,PosTete[i].y,PosTete[i].z);
    }

    //*******************************Inicialización etiquetas de tomas de tiempo
    parciales
    timer.parciales[0].Etiqueta = "Render";
    timer.parciales[1].Etiqueta = "Update";             //Solo en modo Continuo
    timer.parciales[2].Etiqueta = "Gdesk";              //Solo en modo Continuo
    timerIdle.parciales[0].Etiqueta = "Idle";
    timerUpdt.parciales[0].Etiqueta = "UpdateTetera";   //Solo en modo Discreto
    timerUpdt.parciales[1].Etiqueta = "UpdateLuz";      //Solo en modo Discreto
    timer.SetFrequencyHighRes();                        //HRTimer Toma valor de frecuencia
     CPU y toma tick inicial
    timerUpdt.SetFrequencyHighRes();
    timerIdle.SetFrequencyHighRes();
    printf("Frecuencia Timers = %d\n", timer.frecHighRes);

    //*******************************Inicialización GDESK
#ifdef DEF_DISCRETO
    int aux = 1;
    aux = GDESK_StartUpEngine();
    printf("GDESK_StartUpEngine devuelve %d\n", aux);

    //Inicializo timer para controlar el tiempo transcurrido que ha pasado antes de llamar
     al dispatcher
    tmpGdesk.SetFrequencyHighRes();

    //***********************Definimos los tiempos de envio de mensajes GDESK
    //16.67->60hz, 40->25hz , 33,33->30hz
    for(i=0;i<DEF_NUM_TETERAS;i++){
        tete[i].timeGdeskMsg = DEF_MS_UPDATE_T;
    }
    luz.timeGdeskMsg = DEF_MS_UPDATE_L;
    render.timeGdeskMsg = DEF_MS_RENDER;

    //Envio de los primeros mensanjes de GDESK
    for(i=0;i<DEF_NUM_TETERAS;i++){
        tete[i].msg = tete[i].GDESK_GetMessageToFill();
        tete[i].GDESK_SendMessage(tete[i].msg,tete[i],tete[i].timeGdeskMsg);
    }
    luz.msg = luz.GDESK_GetMessageToFill();
    luz.GDESK_SendMessage(luz.msg,luz,luz.timeGdeskMsg);
    render.msg = render.GDESK_GetMessageToFill();
    render.GDESK_SendMessage(render.msg,render,render.timeGdeskMsg);
#endif
}
```

10 Anexo B – Salidas .csv

En la siguientes imágenes se observa un ejemplo de cada una de las salidas en formato csv. Se puede apreciar que contienen siete lineas y cada una corresponde a siete ejecuciones diferentes:

ResumenC.csv

```
1; 54353.847122; 15812.442058; 38463.710362; 2.362290; 0.004346; 75.332412; 126.86
3; 54351.789176; 17583.363337; 36686.163711; 1.121816; 0.002064; 81.140312; 61.86
5; 54366.477403; 18072.116419; 36246.744265; 0.826530; 0.001520; 46.790189; 41.18
10; 54362.310787; 18612.083548; 35709.733622; 0.467838; 0.000861; 40.025779; 22.29
15; 54401.328426; 18827.658810; 35494.099936; 0.361409; 0.000664; 79.208271; 15.26
20; 54379.821442; 18856.565160; 35482.398992; 0.275812; 0.000507; 40.581478; 11.64
25; 54368.024938; 18914.175415; 35396.142348; 0.262678; 0.000483; 57.444497; 9.35
```

ResumenD.csv

```
1; 54368.699888; 3447.701508; 7254.667907; 43631.000000; 80.250218; 35.330474; 24.99
3; 54376.996326; 7298.183561; 14474.449923; 32579.000000; 59.913203; 25.362842; 24.99
5; 54347.675965; 11166.878467; 21725.254251; 21404.000000; 39.383469; 51.543247; 24.98
10; 54400.769274; 20684.846308; 31940.102516; 1750.000000; 3.216866; 25.820450; 24.99
15; 54677.929364; 30549.635969; 23408.410837; 693.000000; 1.267422; 26.882557; 25.00
20; 54401.407845; 40140.460206; 11471.839029; 2761.000000; 5.075236; 28.108610; 24.99
25; 54393.937020; 49862.146159; 18.807041; 4439.000000; 8.160836; 73.983819; 24.98
```

El formato de cada linea corresponde a los siguientes valores:

nº teteras, Ttotal, Trender, Tupdate, Tidle, % CPU libre, TnoControlado, med fps

Para el desarrollo de este trabajo, estos archivos luego han sido utilizados por desde unas Hojas Excel donde registrar los datos y desde un programa en R para la creación de gráficas en formato vectorial.

11 Anexo C – Guía integración

En este Anexo se expone un guía practica de como integrar RT-DESK en una AGTR desarrollada con Glut/OpenGl.

Partiendo del supuesto que todos los objetos de la aplicación están definidos por clases, los pasos a seguir son:

1. Hacer un include de *Entity.h*

2. Hacer que todas las clases hereden de la entidad básica de RT-DESK (*_Entity*). De esta forma todos los objetos heredan funciones y atributos necesarios para la utilización de mensajes.

3. Crear una clase Render que tendrá su propio método para dibujar la escena. Esta clase también debe heredar de la entidad de RT-DESK.

4. Programar el método *_ReciveMessage(_MESSAGE *pMsg)*. Este método es el encargado de tratar los mensajes de eventos que llegan a cada objeto. Todos los objetos tienen que definir que tareas han de realizar en función del tipo de evento que le llegue en el mensaje.

5. En la inicialización del programa hay que añadir la inicialización de RT-DESK a través de la función *_StartUpEngine()*. Esta función devolverá un 0 si la inicialización es correcta.

6. También en la inicialización es importante lanzar los primeros mensajes de eventos a todos los objetos que necesiten realizar tareas periódicas, ya sean de calculo de simulación, de renderizado, o de lo que sea pertinente según la aplicación. Antes de enviar un mensaje hay que adquirirlo. Para pedirle al núcleo de RT-DESK un mensaje se utiliza la función *_GetMessageToFill()*, ahora ya se puede configurar y enviar. Cuando se envía un mensaje hay que indicar cual es el mensaje que se envía, el objeto destinatario y el tiempo de la siguiente ejecución del evento asociado al mensaje (ver

ejemplo en Anexo A – Código Inicialización).

7. En el método de Glut *OnDisplay()*, básicamente solo tiene que estar la llamada a la función

   ProximaSimulacion = _gMsgDispatcher->DeliveMessage();

   Esta función se llama al Dispatcher del núcleo de RT-DESK para que envíe los mensajes que correspondan en ese instante de tiempo. En *ProximaSimulación* nos indica el tiempo que ha de transcurrir hasta volver a ser llamado.

8. En el método de Glut *OnIdle()*, lo único que es necesario hacer es llevar un control del tiempo transcurrido desde el ultimo valor de *ProximaSimulación*. Cuando haya pasado el tiempo indicado por *ProximaSimulación* llamaremos al método de Glut *glutPostRedisplay();* Este método llama inmediatamente a OnDisplay() que a su vez llama de nuevo al Dispatcher.

Nota: El tiempo que esta la ejecución en *OnIdle()* esperando a que llegue el instante de llamar al Dispatcher, es el tiempo de CPU libre que se comenta en apartados anteriores de este trabajo. Durante esta espera es cuando podemos aprovechar para poder realizar otras tareas y así aprovechar este tiempo. Es aconsejable que antes de lanzar una tarea se controle de antemano que el tiempo que tardara en ejecutarse sea menor que el tiempo que falta para llamar al Dispatcher.

Estos vienen ha ser los pasos básicos para poder poner en marcha una aplicación con RT-DESK y Glut/OpenGl.

12 Anexo D – Manual Demo

Partiendo del Benchmark explicado en el capítulo 4 , se han realizado una serie de modificaciones para implementar un demostrador que facilite el entendimiento de la propuesta RT-DESK a través de la interacción con los usuarios que puedan estar interesados en esta tecnología y así facilitar su divulgación.

La Demo implementada muestra en todo momento las lecturas de tiempos. El tiempo de render se muestra mediante los frames por segundo, el tiempo de idle se muestra indicando el porcentaje de CPU libre y el tiempo de update se muestra de forma explicita. Ademas, la Demo ofrece durante la ejecución las siguientes posibilidades:

- Cambio del modo continuo acoplado al modo discreto desacoplado y viceversa (tecla "m").
- Permite añadir o quitar teteras con un mínimo de una y un máximo de 25 (teclas "+" y "-").
- Activar y desactivar proyección de sombras (tecla "s").
- Fijar/Reanudar la luz en cualquier posición (tecla "p") y la rotación de las teteras (tecla "P").
- En modo discreto, cambiar la frecuencia del renderizado (tecla "1").
  Los posibles valores son:
  - 40.00 ms → 25 fps
  - 33.33 ms → 30 fps
  - 16.67 ms → 60 fps
  - 200 ms → 5 fps
  - 100 ms → 10 fps
- En modo discreto, cambiar el tiempo de update de la tetera 1 o tetera central (tecla "2").
  Los posibles valores son:
  40.00 , 33.33 , 16.67 , 100 , 200 , 1 , 0.1 (ms)
- En modo discreto, cambiar el tiempo de update de todas las teteras excepto la tetera central (tecla"3").
  Los posibles valores son:

40.00 , 33.33 , 16.67 , 100 , 200 (ms)

- Por ultimo la opción de finalizar la ejecución (tecla “q” o “Q”).

Ademas se ha tenido que crear una función que reinicia todos los contadores la cual se ejecuta cada vez que se muestra un cambio de las lecturas de tiempos y cada vez que se toca una tecla de control de las mencionadas antes.

También se deben de activar las definiciones de control DEF_CONTINUO y DEF_DISCRETO a la vez, desactivar DEF_MAX_RENDER y DEF_MAX_TIME para que no finalice la ejecución de forma involuntaria, DEF_SOMBRAS ha de estar activada para poder usar la tecla “s”, el valor de DEF_NUM_TETERAS sera el limite de teteras que se pueden crear (valor por defecto 25). El resto de valores no tienen relevancia.

La Figuras 17 y 16 se muestran unas capturas de la Demo en ejecución. En ellas se pueden ver en color rojo la medición de frames por segundo, en color azul la medición del porcentaje de CPU que queda libre y color amarillo la medición del tiempo de update. La “D” (en color azul) indica que el modo discreto esta activado, si estuviera activado el modo continuo seria una “C”.

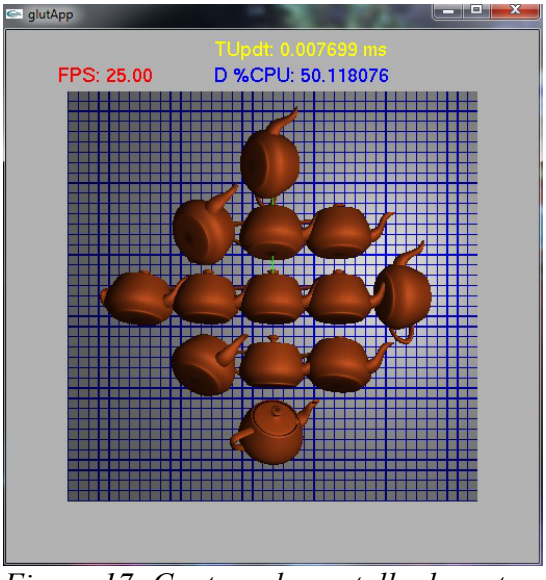

Figura 17: Captura de pantalla durante la ejecución de la Demo

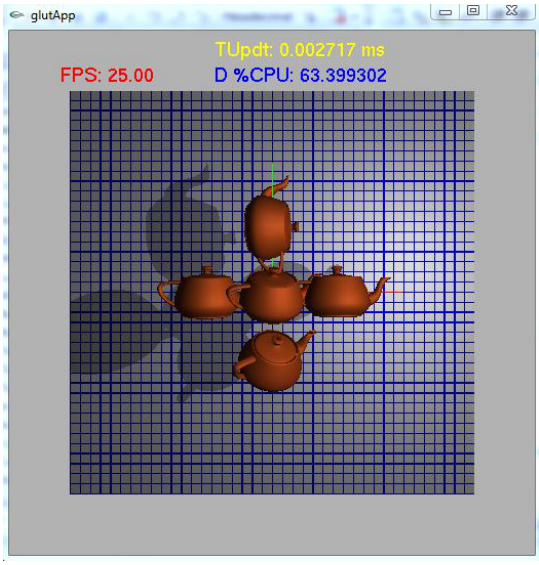

Figura 16: Captura con sombras durante la ejecución de la Demo

La diferencia entre las figuras 17 y 16 es que en la primera no están las sombras activas mientras que el segunda si lo están. En la 17 se puede apreciar que el tiempo de update empleado es superior debido al numero de teteras que se están simulando al mismo tiempo y cada tetera necesita su propia fase de simulación, mientras que en la figura 16 hay menos teteras realizando la fase de simulación. El tiempo de render empleado es muy parecido en las dos ya que en la segunda hay un incremento en la carga de la fase de visualización debido a las sombras proyectadas, en la primera la carga es obviamente generada por el elevado numero de teteras en ejecución.

En la Figura 18 se muestra un captura de los mensajes que aparecen en la Demo cuando se van tecleando las distintas opciones de la Demo cuando se teclean las teclas de control.

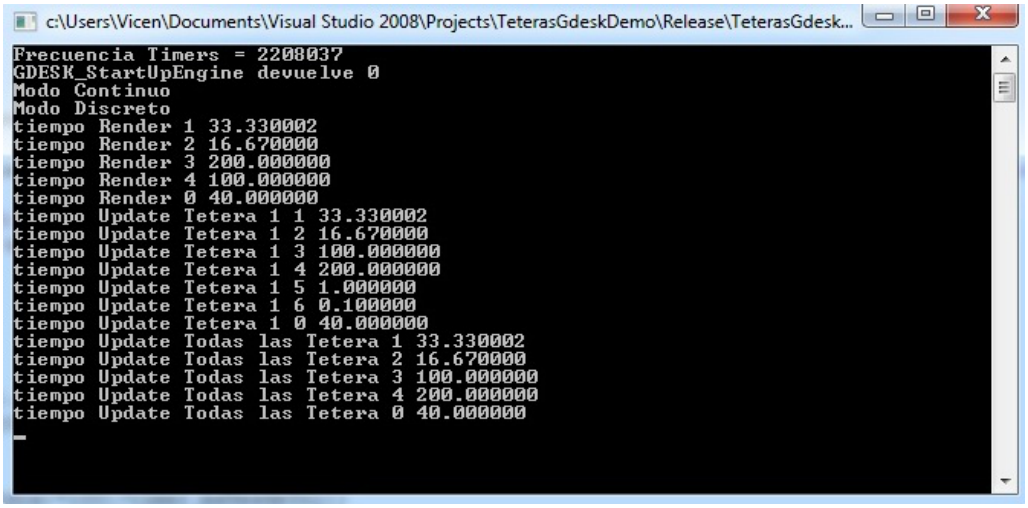

Figura 18: Captura de los mensajes que genera la Demo