novática
CEPIS UPGRADE Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE ***E**uropean **NET**work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2, ASTIC, RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330992 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 – ISSN: 0211-2124; CODEN NOVAEC
Portada: "Salida de la habitación 101" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 190, noviembre-diciembre 2007, año XXXIII

sumario

Monografía del próximo número: "*Gobierno de las TIC*"

Optimización del proceso de *render* 3D distribuido con software libre

Carlos González Morcillo[1,2] Gerhard Weiss[2] David Vallejo Fernández[1] Luis Jiménez Linares[1] Javier Albusac Jiménez[1]

[1] Escuela Superior de Informática, Universidad de Castilla-La Mancha; [2] Software Competence Center, Hagenberg (Austria)

<{carlos.gonzalez, david.vallejo, luis.jimenez, javieralonso.albusac}@uclm.es>,
<{carlos.morcillo, gerhard.weiss}@scch.at>

Resumen: *en la última década las empresas que trabajan con contenido multimedia demandan imágenes de alta definición para sus proyectos de síntesis 3D. El proceso de renderizado consiste básicamente en la obtención de una imagen 2D a partir de la definición abstracta de una escena 3D. A pesar de la aparición de nuevas técnicas y algoritmos, el coste computacional de este proceso es elevado ya que necesita gran cantidad de tiempo para ser realizado, especialmente cuando la escena es compleja o bien cuando es necesario obtener imágenes fotorrealistas. En este artículo se describe Yafrid (Yeah! a Free Render grID) y MAgArRo (MultiAgent AppRoach to Rendering Optimization) ambas arquitecturas han sido desarrolladas en la Universidad de Castilla-La Mancha para la optimización del renderizado distribuido.*

Palabras clave: *agentes inteligentes, inteligencia artificial, optimización, renderizado.*

Autores

Carlos Gonzalez Morcillo es profesor ayudante y estudiante de doctorado en el grupo de investigación ORETO de la Universidad de Castilla-La Mancha. Sus temas de investigación recientes son los sistemas multiagente, *rendering* distribuido y lógica difusa. Obtuvo su licenciatura y un Máster en Informática por la Universidad de Castilla-La Mancha en 2002 y 2004, respectivamente.

.

Gerhard Weiss es el director científico de SCCH (*Software Competence Center Hagenberg GmbH*), uno de los centros de investigación independientes más importantes de Austria. Sus intereses principales son la inteligencia computacional y los sistemas autónomos en general, así como los fundamentos y aplicaciones de la tecnología de agentes y multiagentes en particular. Ha sido coeditor del libro de referencia en esta área "Multiagent Systems" (MIT Press).

David Vallejo Fernandez es profesor ayudante y estudiante de doctorado en el grupo de investigación ORETO de la Universidad de Castilla-La Mancha. Sus temas de investigación recientes son los sistemas multiagente, arquitecturas de inspiración cognitiva y rendering distribuido. Se licenció en Informática por la Universidad de Castilla-La Mancha en 2006.

Luis Jimenez Linares es profesor asociado de Informática en la Universidad de Castilla-La Mancha. Sus temas de investigación más recientes son los sistemas multiagente, representación del conocimiento, diseño de ontologías y lógica difusa. Obtuvo un Máster y un Doctorado en Informática por la Universidad de Granada en 1991 y 1997, respectivamente. Es miembro de la *European Society of Fuzzy Logic and Technology*.

Javier Albusac Jimenez es investigador y estudiante de doctorado en el grupo de investigación ORETO de la Universidad de Castilla-La Mancha. Sus temas de investigación más recientes son los sistemas multiagente, la vigilancia cognitiva y la visión cognitiva. Se licenció en Informática por la Universidad de Castilla-La Mancha en 2006.

1. Introducción

El proceso de *renderizado*[1] consiste en la generación de una imagen 2D a partir de la descripción abstracta de una escena 3D. La construcción de una imagen bidimensional requiere varias fases tales como el modelado, la configuración de materiales y texturas, la ubicación de las fuentes de luz virtuales, y finalmente, el *renderizado*. Los algoritmos utilizados en el proceso de *renderizado* utilizan como entrada información relacionada con la geometría de cada objeto, materiales, texturas, fuentes de luz y cámaras virtuales, y a partir de ésta se obtiene como salida una imagen, o una secuencia de imágenes si se trata de una animación. El *renderizado* de imágenes fotorrealistas de alta calidad es uno de los objetivos principales de la informática gráfica. Desafortunadamente, este proceso es computacionalmente muy costoso y necesita gran cantidad de tiempo cuando las técnicas de *renderizado* empleadas simulan la iluminación global de la escena. Dependiendo del método de *renderizado* y las características de la escena, la generación de una imagen en alta calidad podría suponer varias horas (¡o incluso días!) de cálculo.

En la última década se han propuesto varias soluciones a este problema basadas en el procesamiento paralelo y distribuido. Una de las más populares es las *granjas de render* formadas por ordenadores de una misma organización donde cada uno de los fotogramas de una animación se calcula con un procesador diferente. A las anteriores hay que añadir las propuestas basadas en sistemas *GRID*[2] donde se utiliza Internet para compartir ciclos de CPU.

En este artículo se describe el flujo de trabajo y las herramientas de software libre utilizadas en la Universidad de Castilla-La Mancha (UCLM) en varios proyectos para el *renderizado* de escenas 3D, así como algunas herramientas desarrolladas en la UCLM y distribuidas bajo licencia GPL (*General Public License*). En las próximas secciones se describirá con mayor detalle la arquitectura general de Yafrid y el sistema de optimización del proceso de *render* llamado MAgArRo. Finalmente, se muestran los resultados experimentales y los beneficios que se obtienen de la utilización de un sistema de estas características. El artículo está organizado de la siguiente forma: la siguiente sección ofrece una visión general sobre el estado del arte y algunos de los principales trabajos relacionados con la optimización del proceso de *renderizado*; principalmente, los que están basados en el proceso de *renderizado* en paralelo y distribuido. Las siguientes secciones describen la arquitectura general de un *cluster* basado en Oscar, el sistema *GRID* de *renderizado* llamado Yafrid

y la arquitectura para la optimización inteligente y distribuida del proceso de *renderizado*, llamada MAgArRO. Posteriormente, en la siguiente sección se muestran los resultados que han sido obtenidos empíricamente a partir de los sistemas citados anteriormente. Para finalizar, se expondrán las conclusiones y las propuestas de trabajo futuro.

1.1. Trabajos relacionados

Existen numerosos métodos y algoritmos con diferentes características y propiedades para el proceso de *renderizado* [11][6][10]. Sin embargo, según Kajiya [6] todos los algoritmos de *render* se centran principalmente en el modelo de comportamiento de la luz sobre varios tipos de superficie, y cómo intentan resolver la llamada *ecuación de render*, la cual constituye la base matemáticas de cualquier algoritmo de *render*. Para cualquiera de estos algoritmos, los niveles de realismo alcanzados están relacionados con la complejidad de la escena y el coste computacional requerido. Chalmers et al. exponen en [3] varias líneas de investigación relacionadas con la optimización del *renderizado* distribuido.

Optimización mediante hardware. Una alternativa para reducir el tiempo de *renderizado* consiste en realizar optimizaciones particulares mediante hardware. En esta línea de investigación existen diferentes aproximaciones; algunos métodos utilizan GPUs (*Graphics Processing Units*) programables como sistemas de paralelización masiva. De esta forma se dispone de potentes procesadores para la ejecución en paralelo de diferentes fragmentos de código de un trazador de rayos. El uso de GPUs programables superan a las CPUs (*Central Processing Unit*) de una estación de trabajo estándar en un factor de siete [2]. El uso de la CPU en conjunción con la GPU requiere nuevos paradigmas y alternativas a las arquitecturas tradicionales. Por ejemplo, la arquitectura propuesta por Rajagopalan et al. [8] muestra el uso de un GPU para *renderizar* imágenes en tiempo real a partir de conjuntos de datos complejos, los cuales requieren tareas de cómputo de gran complejidad. Existen algunos motores de *render* diseñados específicamente para ser utilizados con aceleración GPU, tales como Parthenon Renderer [5] el cual utiliza el punto flotante de la GPU, o el motor de *render* Gelato que trabaja con tarjetas gráficas Nvidia.

Optimización utilizando computación distribuida. Si dividimos el problema en otros más pequeños (y cada uno de ellos es resuelto en un procesador diferente), el tiempo necesario para resolver el problema completo debería disminuir. Para obtener una solución óptima al problema todos los ordenadores que forman el sistema deberán mantenerse ocupados. Por tanto, es necesario elegir una buena estrategia de distribución de tareas para conseguir este objetivo.

De acuerdo a la forma en la que se ha realizado la división de tareas, podemos distinguir sistemas de **granularidad fina**, si una imagen se ha dividido en fragmentos más pequeños y éstos han sido enviados a distintos procesadores para que sean ejecutados de forma independiente, o bien, sistemas de **granularidad gruesa** (en el caso de las animaciones) si cada fotograma de la animación sin fragmentar es enviado a una unidad de procesamiento. Dr. Queue [17] es una herramienta de código abierto diseñada para distribuir fotogramas a través de una granja de computadores interconectados en una red; en concreto, este software multi-plataforma trabaja en un nivel de división de granularidad gruesa. En la **sección 2**, la solución que proponemos está basada en Oscar [18], una herramienta de código abierto para la construcción de *clusters*. Muchas de las nuevas propuestas que se hacen en la literatura para el *rendering* distribuido están basadas en sistemas *GRID*, donde las tareas se distribuyen entre un número amplio de ordenadores conectados a Internet con características heterogéneas; estos sistemas se basan principalmente en el aprovechamiento de los ciclos de CPU de aquellos procesadores que se encuentran ociosos [1]. Esta tecnología emergente se denomina *Volunteer Computing* o *Peer-to-peer Computing*, y actualmente, se utiliza en algunos proyectos basados en la tecnología BOINC (*Berkeley Open Infraestructure for Network Computing*: por ejemplo, BURP [16] *Big and Ugly Rendering Project*). En la **sección 3**, se describirá la arquitectura general de Yafrid y las ventajas clave derivadas de su utilización.

Predicción del coste. El conocimiento relativo a la distribución de costes en la escena (es decir, en los diferentes fragmentos de la escena dividida) puede facilitar significativamente la localización de recursos cuando se está haciendo uso de una aproximación distribuida. De hecho, esta estimación cobra mayor importancia cuando hablamos de producción comercial de *renderizado* (una predicción fiable del coste podría ayudar a cumplir los plazos establecidos y a poder elaborar los presupuestos con mayor precisión). Para solucionar el problema anterior, existen numerosas propuestas basadas en el conocimiento sobre la distribución de costes; un buen ejemplo es [9]. En la **sección 4** exponemos el mecanismo para la predicción de costes utilizado en MAgArRO.

Optimización con sistemas multi-agente distribuidos. Los sistemas multi-agente distribuidos nos permiten obtener una posible alternativa a las propuestas anteriores para la optimización del proceso de *renderizado*. El trabajo de Rangel-Kuoppa [7] utiliza una plataforma multi-agente, basada en JADE, para la distribución interactiva de tareas de *rendering* en un red. Aunque este trabajo emplea la metáfora de sistema multi-agente, en realidad no hace uso de una tecnología multi-agente como tal. De hecho, la plataforma JADE se utiliza únicamente con el propósito de comunicar los nodos, sin utilizar ningún conocimiento experto ni negociación. La arquitectura de MagArRo propuesta en la **sección 4** es un ejemplo de una arquitectura multi-agente distribuida, que emplea el conocimiento proveniente de los expertos para optimizar los parámetros de *renderizado*.

2. *Cluster* basado en Oscar

Hoy en día, las Universidades disponen de laboratorios acondicionados con gran cantidad de ordenadores para las clases prácticas. Sin embargo, la mayoría de ellos se encuentran inactivos durante el periodo vacacional, fines de semana y durante las noches. Esta infraestructura hardware podría ser utilizada y coordinada durante los intervalos de tiempo en los que no es necesario el uso de los ordenadores [14]. Oscar [18] es una plataforma software que permite la programación de *clusters* con máquinas basadas en GNU/Linux. A continuación, se explicará en detalle la arquitectura general del sistema basada en Oscar, y que ha sido utilizada por la Universidad de Castilla-La Mancha para el *renderizado* de algunos proyectos 3D [20][22].

2.1. Esquema general de la arquitectura

En nuestro entorno de ejecución, el sistema está compuesto por 130 estaciones de trabajo localizadas en diferentes aulas. Estos computadores son PCs con características diferentes. Cada una de las aulas tiene un tipo de hardware específico (todos ellos basados en arquitectura x86). Los requerimientos mínimos que debe tener cada computador para pertenecer al sistema son una memoria RAM de 500 MB, una partición SWAP de 1 GB, y tener una conexión de al menos 100Mbits/s (todos los ordenadores están conectados a la red utilizando switches de 100Mbits/s). En la **figura 1** se muestran estos requisitos.

Las aulas donde se utiliza el *cluster* basado en Oscar están dedicadas a una actividad educacional. Por esta razón, no es apropiado instalar software en explotación (de forma permanente) en las máquinas. El subproyecto Thin-Oscar [19] nos permite utilizar máquinas sin disco duro local o partición dedicada para instalar el sistema operativo como miembro del *cluster* Oscar. La

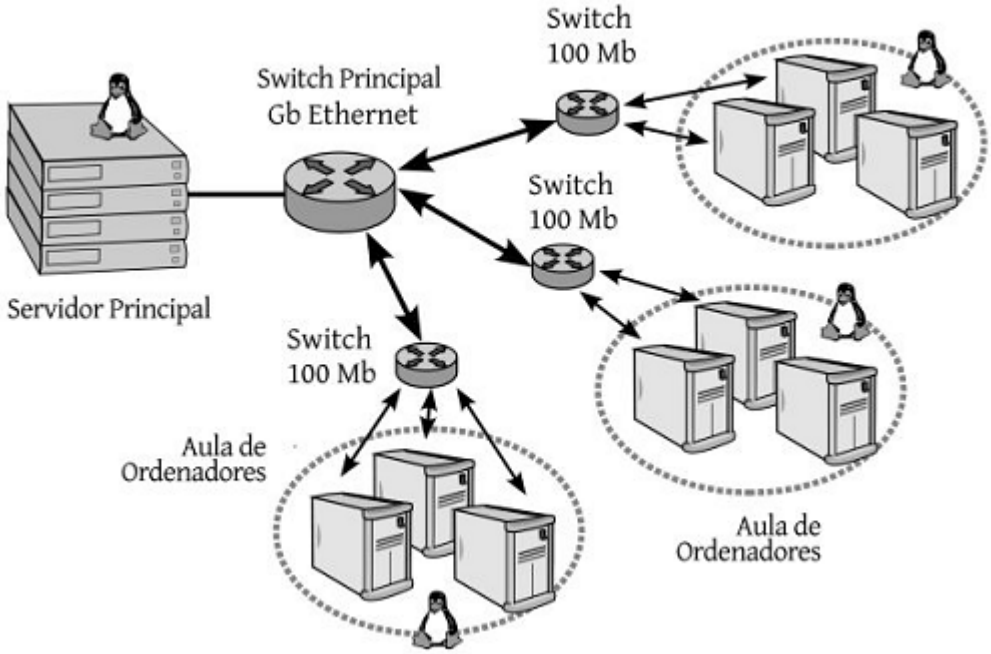

Figura 1. Granja de *rendering* basada en Oscar ubicada en ESI-UCLM.

partición SWAP se utiliza para gestionar los archivos temporales (utilizados en el proceso de *renderizado* en cada proyecto).

Cada nodo del sistema distribuido es configurado para obtener los parámetros de configuración de la red; para ello se utiliza la extensión PXE de la BIOS (*Pre eXecution Environment*). En nuestro caso estos datos se encuentran en la imagen del sistema operativo que será ejecutado. De esta forma, sin necesidad de instalar ningún tipo de software, se carga la imagen del sistema operativo (que está formada únicamente por los servicios básicos y el motor de *render*).

El servidor tiene dos procesos clave para atender las peticiones PXE:

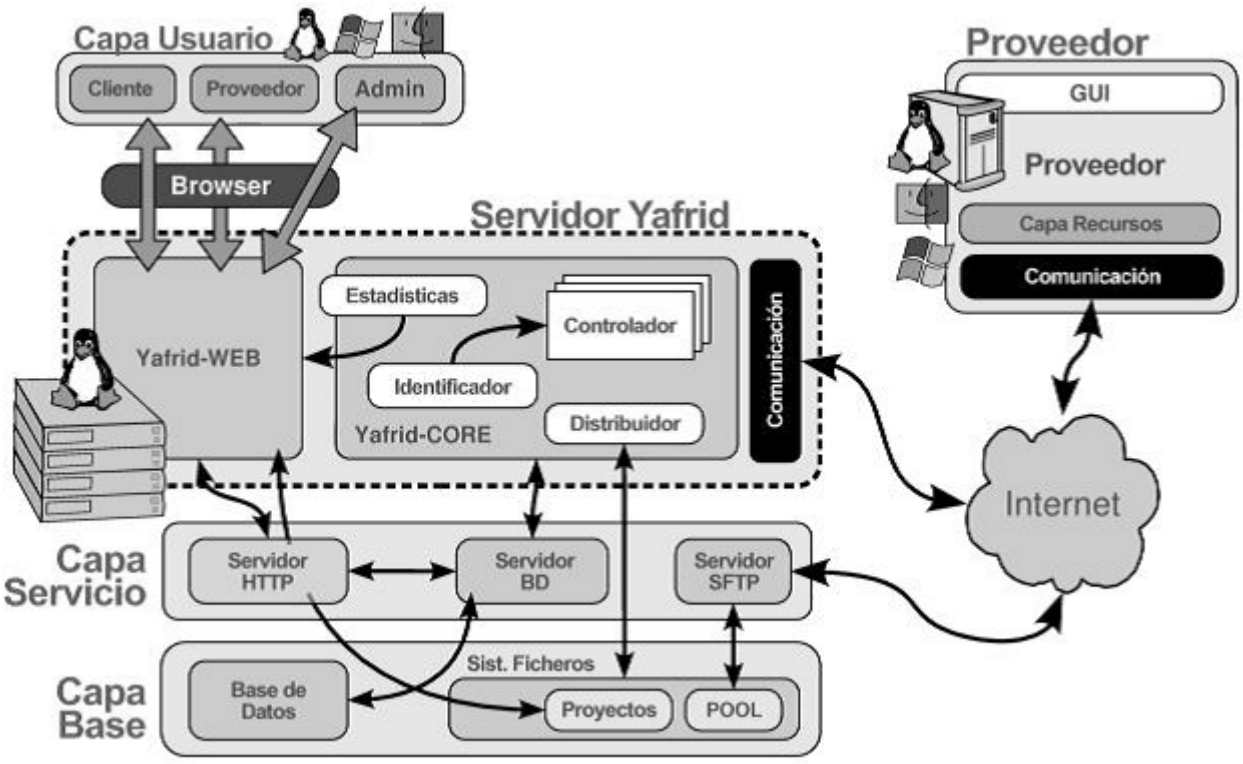

Figura 2. Arquitectura general de Yafrid.

■ **DHCPD** (*Dynamic Host Configuration Protocol Daemon*): este protocolo se utiliza para informar a cada cliente de la IP que le corresponde y la imagen del sistema operativo que debe cargar.
■ **TFTPD** (*Trivial Transfer Protocol Daemon*): Cuando el servidor recibe una petición de archivo, éste consulta las tablas de configuración para enviarlo al cliente.

Para activar o desactivar los ordenadores del sistema, se hace uso de una extensión de las BIOS actuales llamada WOL (*Wake On Lan*). Esta extensión se utiliza con la ayuda de la placa madre y el paquete software Ether-Wake (desarrollado por Donal Becker). Incluso si un ordenador se encuentra apagado, la tarjeta de red continúa escuchando posibles peticiones. Cuando el paquete generado por Ether-Wake llega a la tarjeta, el ordenador se enciende y por medio de una petición PXE se carga la imagen del sistema operativo. Finalmente, para apagar por completo los ordenadores es necesario configurar correctamente la interfaz ACPI (*Advanced Configuration and Power Interface*). Para poder hacer uso de esta funcionalidad se implementó un sencillo script en el lado del servidor el cual establece una conexión ssh con cada nodo y les envía el comando *shutdown* para apagarlos.

3. Yafrid: sistema de *renderizado* basado en *Grid*

Yafrid es un sistema que aprovecha las ventajas de los sistemas de computación *Grid*, permitiendo la distribución de una escena a través de Internet para su *renderizado*. Además, el sistema soporta otras tareas importantes como son la gestión de las unidades de trabajo y el uso controlado del *grid*.

3.1. Esquema general de la arquitectura

Los componentes de Yafrid que se encuentran en la capa de más alto nivel de la arquitectura son:
■ **Servidor**. Es el núcleo de Yafrid. Su componente básico es el *Distribuidor*, encargado de recoger los trabajos de una cola y enviarlos a los proveedores.
■ **Proveedor** . Esta entidad se encarga de procesar las peticiones de los clientes.
■ **Cliente**. Un cliente es una entidad externa que no pertenece al sistema en un sentido estricto. La función principal del cliente es añadir trabajos al sistema *grid*, para que posteriormente sean completados por los proveedores.

A continuación se describen los tres roles de usuario utilizados en Yafrid:
■ **Cliente**. Este rol permite a un usuario añadir nuevos trabajos al *grid*. Un cliente también puede crear y gestionar grupos asociados a los proyectos (clientes y proveedores pueden suscribirse a estos grupos). Únicamente los proveedores que pertenecen al mismo grupo pueden participar en el proceso de *renderizado* del proyecto.
■ **Administrador**. Este usuario es necesario para operar con cualquiera de los componentes que constituye el sistema, y además, posee todos los privilegios para acceder a la información relacionada con cualquier usuario del sistema.
■ **Proveedor.** El proveedor es un usuario que tiene instalado el software necesario para que el *grid* pueda enviarle trabajos.

Servidor Yafrid. El servidor es el nodo fundamental ya que todo el sistema gira en torno a él. Cada uno de los proveedores se conecta a este nodo para ceder al *grid* sus ciclos de CPU, con el objetivo de *renderizar* las escenas que los clientes han añadido.

En la **figura 2** se muestran las cuatro capas que forman la arquitectura del servidor Yafrid. Este diseño está basado en la arquitectura de [4]. Las capas mencionadas anteriormente son *Capa Base (o de Recursos*, *Capa de Servicio*, *Servidor Yafrid* y *Capa de usuario*.

Capa de recursos. Ésta es la capa de menor nivel de abstracción. La capa de recursos tiene los siguientes componentes:
■ **Sistema de base de datos**. En las bases de datos de esta capa se encuentran las tablas donde se almacena la información necesaria para el correcto funcionamiento del sistema. Algunas de estas tablas se utilizan para obtener estadísticas de la ejecución del sistema, mientras que otras almacenan datos asociados a usuarios, grupos, proyectos, etc. Los niveles que se encuentran sobre esta capa en la jerarquía pueden acceder a las bases de datos a través del servidor de bases de datos. La implementación actual del sistema utiliza MySQL como servidor para las bases de datos.
■ **Sistema de archivos**. En algunas ocasiones es necesario acceder directamente al sistema de archivos desde las capas superiores. Básicamente el sistema distingue dos tipos de directorios: uno de ellos se utiliza para almacenar las unidades de trabajo de los proyectos creados, y el segundo tipo contiene información relativa a los usuarios y sus proyectos.
■ **Sistema de red**. El módulo dedicado a las comunicaciones, que pertenece a la capa principal, oculta la utilización de los recursos de red de los ordenadores mediante el uso de un *middleware* (la implementación actual utiliza ZeroC ICE).

Capa de Servicio. Esta capa contiene los servidores que permiten a los módulos de las capas superiores acceder a los recursos que pertenecen a las capas inferiores. Los servidores que actúan en esta capa son los siguientes:
■ **Servidor HTTP.** El módulo web de Yafrid trabaja sobre este servidor. Como este módulo ha sido implementado con lenguajes de programación que permiten la construcción de páginas dinámicas (la implementación actual está escrita en lenguaje PHP), el servidor web debe proporcionar soporte para este tipo de lenguajes. También, es necesario tener soporte para la composición dinámica de gráficos y el acceso a la base de datos.
■ **Servidor de bases de datos**. Este servidor es utilizado por los módulos de Yafrid para acceder a los datos que son indispensables para el correcto funcionamiento del sistema.
■ **Servidor SFTP**. Los proveedores utilizan este servicio para obtener los archivos necesarios para el *renderizado* de las unidades de trabajo. Una vez que el proceso de *renderizado* ha finalizado, se utilizará el servidor SFTP (*Simple File Transfer Protocol*) para enviar al servidor de Yafrid la imagen resultante.

Capa Yafrid. Esta es la capa principal del servidor y está compuesta de dos módulos diferentes (Yafrid-WEB y Yafrid-CORE) que trabajan independientemente el uno del otro. **Yafrid-WEB** es el módulo interactivo del servidor y ha sido implementado mediante un conjunto de páginas dinámicas. **Yafrid-CORE** es la parte no interactiva del servidor.

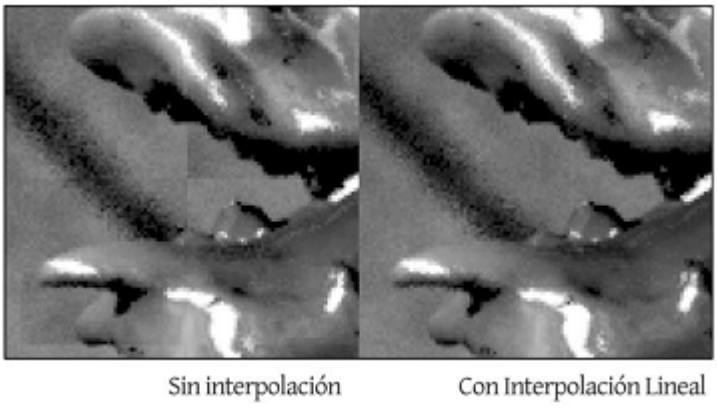

Figura 3. Artefactos sin interpolación entre unidades de trabajo.

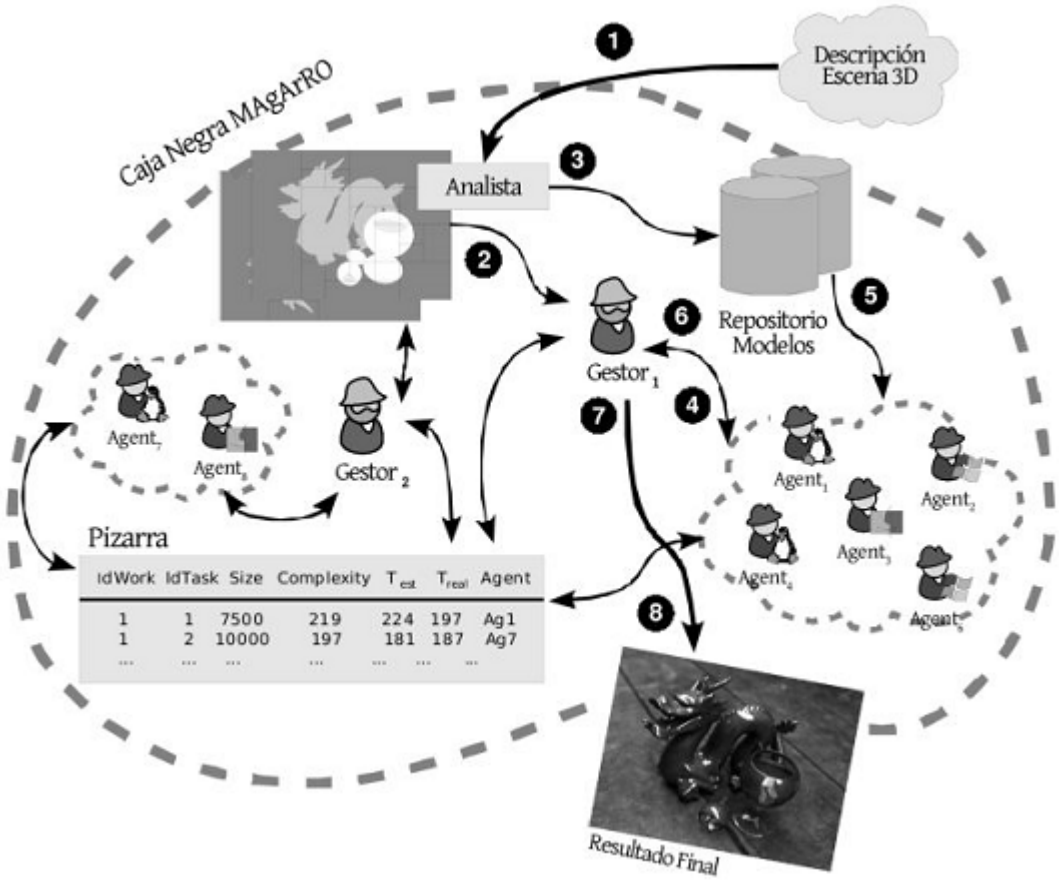

Figura 4. Flujo de trabajo y principales roles en la arquitectura.

Este módulo ha sido principalmente desarrollado utilizando el lenguaje Python. Además, Yafrid-CORE está compuesto por tres submódulos; Distribuidor, Identificador y Estadísticas.

■ **El Distribuidor** es la parte activa del servidor encargado de realizar las siguientes tareas indispensables: generación de unidades de trabajo, asignación de las unidades de trabajo a los proveedores, control del *Timeout*, finalización de proyectos y composición de resultados. A partir de los resultados generados por los proveedores, el distribuidor compone la imagen final. Este proceso no es trivial ya que se pueden distinguir pequeñas diferencias entre los fragmentos obtenidos de diferentes computadores, debido a la componente aleatoria de los métodos basados en Monte Carlo. Por esta razón es necesario suavizar la zona de unión entre los fragmentos mediante una mascara de interpolación lineal. En la **figura 3** (izquierda) podemos apreciar los problemas que surgen al unir dos fragmentos si no se utiliza algún método de combinación de los mismos.

■ La parte pasiva de Yafrid-CORE se denomina módulo **Identificador** cuya misión principal consiste en establecer las comunicaciones de los proveedores. La primera vez que el proveedor intenta conectar con el servidor Yafrid, el Identificador genera un objeto (controlador del proveedor) y le devuelve un *proxy*. Es importante destacar que cada proveedor tiene su propio objeto controlador.

■ **Proveedor**. El proveedor es el software que un usuario debe instalar en su ordenador para que el sistema *grid* pueda utilizar sus ciclos de CPU ociosos. La primera tarea que debe llevar a cabo un proveedor es establecer la conexión con el *grid*. Una vez activado, el proveedor espera hasta que el servidor le envía una unidad de trabajo para procesarla. Cuando finaliza el proceso de *renderizado*, el proveedor envía el archivo vía SFTP e informa al controlador que el trabajo ha finalizado.

4. MagArRO: optimización inteligente y distribuida del proceso de renderizado

De acuerdo a [12] , un agente es un sistema que se encuentra situado en un entorno determinado y es capaz de actuar de forma autónoma en dicho entorno para conseguir los objetivos para los que fue diseñado. MagArRO utiliza los principios, técnicas y conceptos del área a la que pertenecen los sistemas multi-agente, y además, está basado en los principios de diseño de los estándares de FIPA (*Foundation for Intelligent Physical Agents*) [21].

Para el desarrollo de MagArRO también se ha utilizado el *middleware* ICE [25]. IceGrid es el módulo de ICE encargado de la localización de servicios y se utiliza para indicar donde residen los servicios proporcionados por cada ordenador del *grid*. Por otra parte, Glacier2 se utiliza para resolver las dificultades relacionadas con los entornos de red hostiles, haciendo posible que los agentes puedan conectarse a través de un *router* y un *firewall*.

4.1. Esquema general de la arquitectura

En la **figura 4**, se muestra el flujo de trabajo y los principales roles que participan en la arquitectura. Aparte de los servicios básicos de FIPA, MAgArRO incluye servicios específicos relacionados con la optimización del proceso de *renderizado*. En concreto, un servicio denominado *Analista* estudia la escena para dividirla en varios fragmentos. Finalmente, el servicio llamado *Gestor (o Master)* maneja grupos de agentes dinámicos que cooperan para satisfacer las subtareas.

En la **figura 4**, también podemos ver el flujo

de trabajo básico en MagArRO (los círculos numerados lo definen).

1.- El primer paso es la suscripción de los agentes al sistema. Esta suscripción puede hacerse en cualquier momento; los agentes disponibles son gestionados dinámicamente. Cuando el sistema recibe un nuevo archivo para ser *renderizado*, éste es atendido por el Analista. 2.- El Analista analiza la escena, haciendo algunas particiones del trabajo y extrayendo un conjunto de tareas. 3.- Se informa al Gestor de que una nueva escena ha sido creada en el sistema, y además es enviada al Repositorio de Modelos. 4.- Algunos de los agentes disponibles en ese momento son gestionados por el Gestor y se les informa sobre la nueva escena. 5.- Cada uno de los agentes obtiene el modelo 3D del repositorio y comienza un proceso de subasta. 6.- Los agentes ejecutan las (sub) tareas y los resultados obtenidos son enviados al Gestor. 7.- El Gestor compone el resultado final utilizando la salida de las tareas que han sido resueltas previamente. 8.- El Gestor envía la imagen *renderizada* al usuario.

Análisis de la escena utilizando mapas de importancia. MAgArRO utiliza la idea de estimar la complejidad de las diferentes tareas para conseguir el particionamiento de carga balanceada. El agente Analista analiza la complejidad previa e independientemente al resto de pasos pertenecientes al proceso de *renderizado*.

El principal objetivo en este proceso de fragmentación es obtener tareas con complejidad similar para evitar la demora en el tiempo final causada por tareas demasiado complejas. Este análisis se realiza rápidamente y de forma independiente del proceso final de *render*. Una vez que el mapa de importancia es generado, se construye una partición para obtener un conjunto final de tareas. Estas particiones constan de diferentes niveles organizados jerárquicamente, donde en cada nivel se utilizan los resultados del proceso de fragmentación obtenidos del nivel anterior.

En el primer nivel, la partición está hecha siendo cuidadosos con el tamaño mínimo y la complejidad máxima de cada zona. Con estos dos parámetros, el *Analista* realiza una división recursiva de las zonas (ver **figura 5**). En el segundo nivel, se unen las zonas vecinas con complejidad similar. Finalmente, en el tercer nivel el *Analista* intenta obtener una división equilibrada en la que cada zona tenga casi el mismo ratio complejidad/tamaño. La idea que subyace en la división es la de obtener tareas que requieran aproximadamente el mismo tiempo de *renderizado*. Como se muestra abajo en los resultados experimentales, la calidad de este particionamiento está directamente relacionada con el tiempo final de *renderizado*.

Utilizando el Conocimiento Experto. Cuando una tarea es asignada a un agente, se utiliza un conjunto de reglas difusas para modelar el conocimiento experto y optimizar los parámetros del proceso de *renderizado* para esta tarea. Los conjuntos de reglas difusas son considerados apropiados para el modelado del conocimiento experto debido a su poder descriptivo y facilidad de extensión [13]. Los parámetros de salida (la parte consecuente de las reglas) son configurados de modo que el tiempo requerido para completar el *renderizado* se reduzca y la pérdida de calidad esté minimizada. Cada agente puede modelar diferente conocimiento experto con un conjunto distinto de reglas difusas. Por ejemplo, la siguiente regla se usa (en un conjunto de 28 reglas) para describir los parámetros de *renderizado* del método Pathtracing:

R_1: **If** C is {B, VB} $\subset$ S is B,N $\subset$ Op is VB **then** Ls is VS $\subset$ Rl is VS

El significado de esta regla es "Si la complejidad es grande o muy grande y el tamaño es grande o normal y el nivel de optimización es muy grande entonces el número de muestras de luz es muy pequeño y el nivel de recursión es muy pequeño". El parámetro de complejidad representa el ratio complejidad/tamaño de una tarea, el tamaño es el de una tarea en píxeles y el nivel de optimización es seleccionado por el usuario. El parámetro de salida *nivel de recursión*, define el nivel global de recursión en el trazado de rayos (número de rebotes de luz) y las muestras de luz definen el número de muestras por luz en la escena (si es más grande, tendremos mayor calidad y mayor tiempo de *renderizado*).

5. Resultados experimentales

Para analizar el comportamiento de los sistemas se conectaron a Yafrid y MagArRO 8 ordenadores con las mismas características. Estos nodos (Intel Pentium Centrino 2 Ghz, 1GB RAM) se usaron en ambos sistemas durante la ejecución de todas las pruebas. La escena a analizar contenía más de 100.000 caras, 5 niveles de recursión de trazado de rayos en superficies espejo (dragón), 6 niveles en superficies transparentes (copa), y 128 muestras por fuentes de luz fueron *renderizadas* usando el motor libre de *renderizado* Yafray [23]. Además , 200.000 fotones fueron lanzados para construir la estructura del mapa de fotones. Con esta configuración, el *renderizado* en una sola máquina sin ningún tipo de optimización necesitó 121:17 (121 minutos y 17 segundos).

En el caso de Yafrid, como se puede obervar en la **figura 6** (*izquierda*), el tiempo de *renderizado* usando la rejilla es, en el mejor de los casos casi siete veces mejor y dos veces menor en el peor de los casos. Con estos resultados queda clara la importancia de elegir un apropiado tamaño de unidad de trabajo. Esto ocurre debido a que existen tareas complejas que ralentizan el proceso completo de *renderizado* incluso si se aumenta el número de nodos.

Como se ha mencionado anteriormente, MAgArRO usa Mapas de Importancia para estimar la complejidad de las diferentes tareas. La **figura 6** (*derecha*) muestra el tiempo requerido usando diferentes niveles de partición. Utilizando un simple particionamiento de primer nivel (similar al enfoque de Yafrid), puede obtenerse un buen nivel de *renderizado* con pocos agentes. Sin embargo, cuando el número de agentes (nodos de procesamiento) crece, el rendimiento total del sistema se incrementa porque las diferencias en la complejidad de las tareas son relativamente pequeñas.

Figura 5. Mapas de importancia. *Izquierda*: Particionamiento a ciegas (Primer Nivel). *Centro:* Zonas de unión con complejidad similar (Segundo Nivel). *Derecha:* Equilibrio ratio complejidad/tamaño (Tercer Nivel).

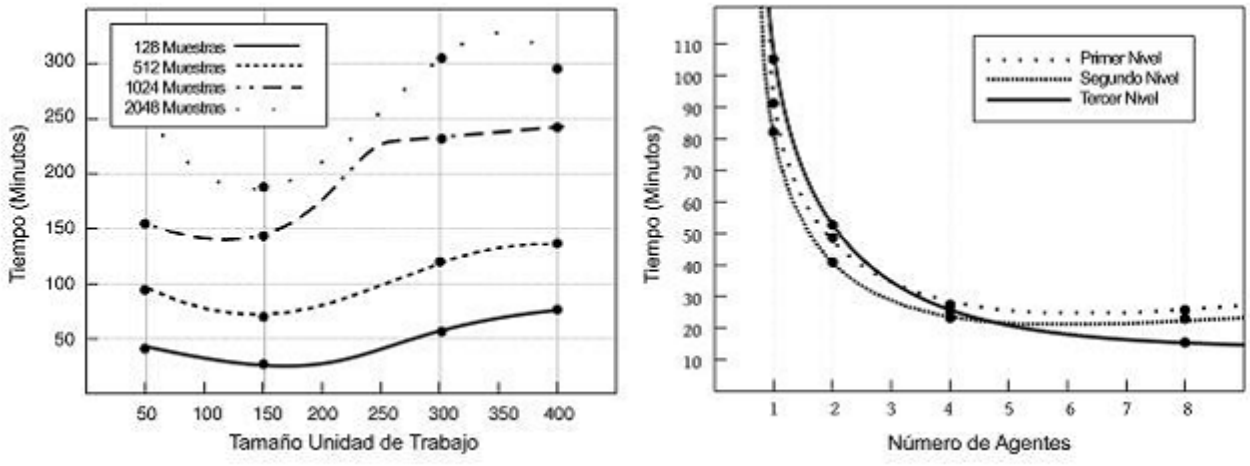

Figura 6. *Izquierda*: Yafrid. Tiempo de *renderizado* relacionado con el tamaño de una unidad de trabajo. *Derecha*: MagArRO. Diferentes niveles de particionamiento con un nivel normal de optimización.

Como observación final, apuntar que el proceso de optimización inteligente puede dar lugar a diferentes niveles de calidad para diferentes áreas en la escena total. Esto es debido a que niveles más agresivos de optimización (Grande o Muy Grande) pueden provocar pérdida del detalle.

Por ejemplo, en la Figura 7.e, los reflejos en el vaso no están tan definidos como en la figura 7.a. La diferencia entre el *renderizado* óptimo y el nivel más agresivo de optimización (Figura 7.f) es mínimo.

6. Discusión y conclusiones finales

Los requisitos computacionales que son necesarios para el *renderizado* de calidad fotorrealista son enormes y, por tanto, obtener resultados en un tiempo razonable y en un sólo ordenador es prácticamente imposible (incluso existen más dificultades en el caso de las animaciones). Muchas propuestas, basadas en diferentes tecnologías, han sido expuestas en este artículo.

Nuestro *cluster* basado en el sistema **Oscar** presenta algunas características interesantes:

- Muy buen rendimiento en el *renderizado* de animaciones. El sistema divide cada fotograma de la animación en diferentes nodos del *cluster*. El enfoque de granularidad fina requiere la programación de nuevas características en el servidor principal.
- El procesamiento de nodos es usado durante tiempos ociosos (por ejemplo, durante la noche).
- La latencia debida a la transferencia de archivos es mínima (gracias al uso de una red Ethernet de gran velocidad).

Por lo demás, el *cluster* sólo puede utilizarse añadiendo tareas al servidor principal en la misma organización. Para resolver algunos de estos problemas, se diseñó **Yafrid**. Este sistema *grid* tiene algunas ventajas importantes:

- No es un *cluster*, los proveedores pueden ser heterogéneos (software y hardware) y pueden estas distribuidos geográficamente.
- Con la aproximación de granularidad

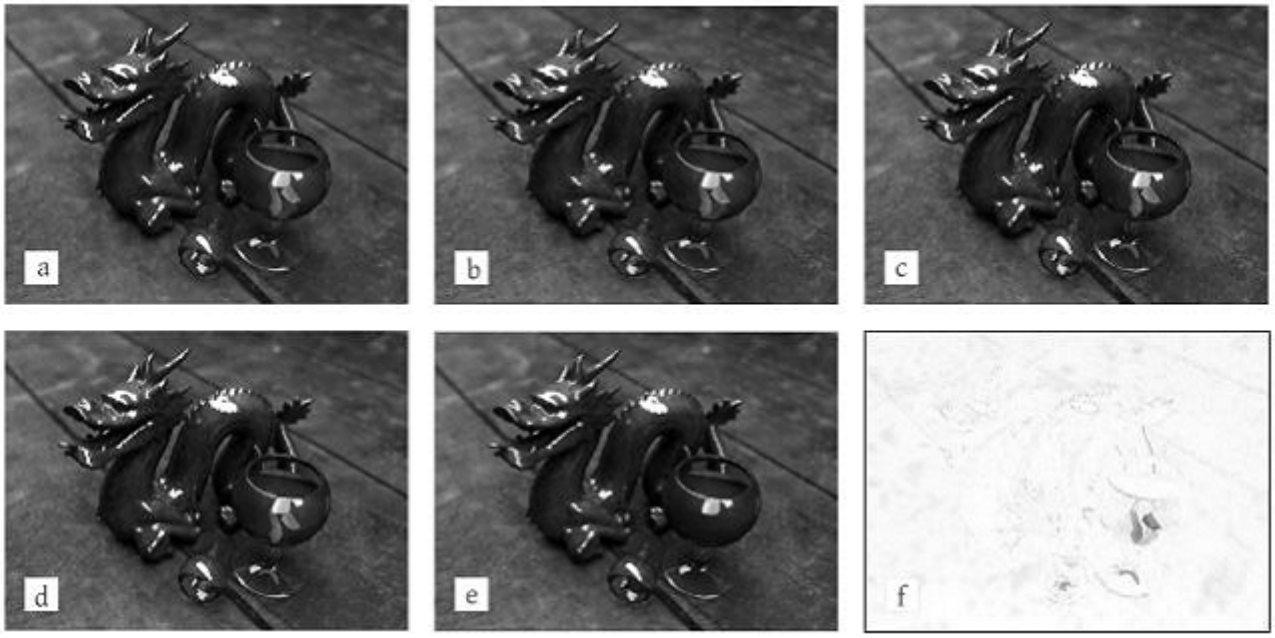

Figura 7. Resultado del *renderizado* utilizando diferentes niveles de optimización. (a) Sin optimización y *renderizado* en una máquina. (b) Muy Pequeño (c) Pequeño (d) Normal (e) Muy Grande (f) Diferencias entre (a) y (e) (la claridad del color, la diferencia más pequeña).

fina, se pueden realizar optimizaciones locales en cada fotograma.

- Una de las principales ventajas de este sistema distribuido es la escalabilidad. La ejecución percibida por el usuario depende del número de proveedores subscritos.

Algunas mejoras podrían realizarse perfeccionando la ejecución de Yafrid. Algunas de ellas se han añadido a **MAgArRO**:

- MAgArRO permite dirigir la importancia del *renderizado* a través del uso de mapas de importancia.
- Permite la aplicación de conocimiento experto mediante el empleo flexible de reglas difusas.
- Aplica los principios de control descentralizado y optimización local. Los servicios son fácilmente replicables, de manera que los cuellos de botella pueden ser minimizados en el despliegue final.

Existen muchas líneas de investigación futura. En nuestro trabajo actual, nos concentramos en la combinación de las mejores características de Yafrid y MAgArRO para integrar el nuevo sistema (llamado YafridNG) en el repositorio oficial de Blender [15]. El código fuente de estos sistemas, distribuidos bajo licencia GPL, pueden ser descargados desde [24].

Agradecimientos

Este trabajo ha sido financiado por la *Consejería de Ciencia y Tecnología* y la *Junta de Comunidades de Castilla la Mancha* bajo los proyectos de investigación PAC06-0141 y PBC06-0064. Agradecimientos especiales a Javier Ayllon por su ayuda al Servicio Supercomputacional (Universidad de Castilla-La Mancha).

Referencias

[1] D.P. Anderson, G. Fedak. The Computational and Storage Potential of Volunteer Computing. *Sixth IEEE International Symposium on Cluster Computer and the Grid (CCGRID '06)*. 7380. Mayo 2006.

[2] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. *Proceedings of SIGGRAPH '04*, 777786.

[3] A. Chalmers, T. Davis, E. Reinhard. *Practical Parallel Rendering*. Ed. A. K. Peters, 2002. ISBN: 1-56881-179-9.

[4] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications 15, 3*(2002).

[5] T. Hachisuka. High-Quality Global Illumination Rendering using Rasterization. *GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[6] J.T. Ka jiya. The rendering equation. *Computer Graphics 20(4): 143150*. Proceedings of SIGGRAPH '86.

[7] R.R. Kuoppa, C.A. Cruz, D. Mould. Distributed 3D Rendering System in a Multi-Agent Platform. *Proceedings of the Fourth Mexican International Conference on Computer Science, 8, 2003*.

[8] R. Ra jagopalan, D. Goswami, S.P. Mudur. Functionality Distribution for Parallel Rendering. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 1828, April 2005.

[9] E. Reinhard, A.J. Kok, F.W. Jansen. Cost Prediction in Ray Tracing. *Rendering Techniques '96, 4150*. Springer-Verlag, June 1996. ISBN 3-211-82883-4.

[10] E. Veach, L.J. Guibas. Metropolis light transport. *Proceedings of SIGGRAPH'97, 6576*. New York, USA: ACM Press - Addison Wesley Publishing Co.

[11] T. Whitted. An improved illumination model for shaded display. *Proceedings of SIGGRAPH '79, 14*. New York, USA: ACM Press.

[12] M.J. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2002. ISBN: 0-471-49691-X

[13] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning. *Information Science*, 1975.

[14] Beowulf. Open Scalable Performance Clusters. <http://www.beowulf.org>.

[15] Blender. Free 3D content creation suite. <http://www.blender.org>.

[16] BURP. Big Ugly Rendering Pro ject. <http://burp.boinc.dk/>.

[17] Dr. Queue. OS Software for Distributed Rendering. <http://www.drqueue.org/>.

[18] OSCAR. Open Cluster Group. <http://www.openclustergroup.org/>.

[19] Thin-Oscar. <http://thin-osccar.sourceforge.net/>.

[20] Virtual Tour ESI UCLM. <http://www.inf-cr.uclm.es/virtual/index_en.html>.

[21] FIPA. Foundation for Intelligent Physical Agents. <http://www.pa.org>.

[22] Virtual Visit - Hospital Ciudad Real. <http://dev.oreto.inf-cr.uclm.es/www/vvhosp>.

[23] Yafray. Yet Another Free Raytracer <http://www.yafray.org>.

[24] Yafrid Next Generation. <http://www.yafridng.org>.

[25]. ZeroC ICE Middleware. <http://www.zeroc.com>.

Notas

[1] "La renderización es el proceso de generar una imagen desde un modelo. Los medios por los que se puede hacer un renderizado van desde lápiz, pluma, plumones o pastel, hasta medios digitales en dos y tres dimensiones. La palabra renderización proviene del inglés render, y no existe un verbo con el mismo significado en español, por lo que es frecuente usar las expresiones renderizar o renderear" <http://es.wikipedia.org/wiki/Renderizar>.

[2] "Grid computing es una tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado" <http://es.wikipedia.org/wiki/Grid_computing>.