REDAG3D: Renderizado Distribuido de un Ambiente Gráfico 3D

Diana L. REYES
Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá D. C., Colombia

Alfonso BARBOSA
Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá D. C., Colombia

César J. BUSTACARA
Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá D. C., Colombia

RESUMEN

Este artículo presenta el proceso para realizar el renderizado distribuido de ambientes gráficos tridimensionales de forma configurable y dinámica. Inicialmente se describen brevemente conceptos relevantes de los sistemas distribuidos, la computación gráfica y el renderizado distribuido, para luego especificar la arquitectura y la implementación de la aplicación que realiza el proceso, denominada REDAG3D. Para llevar a cabo el renderizado de forma distribuida, REDAG3D permite al usuario final seleccionar entre dos mecanismos de comunicación: Sockets y JNDI, dos técnicas de renderizado distribuido: Sort-First y Sort-Last, y cargar una escena en formato OBJ. Finalmente se presentan las pruebas realizadas y las conclusiones obtenidas.

Palabras Claves: Computación Gráfica, Renderizado Distribuido, Mecanismos de Comunicación, Sistemas Distribuidos, Arquitectura de Sistemas.

1. INTRODUCCIÓN

La computación gráfica ha evolucionado en gran medida desde sus orígenes hasta hoy, gracias al aporte de otras líneas de investigación, tales como los sistemas distribuidos y los mecanismos de comunicación. Actualmente existen varias arquitecturas que permiten realizar el renderizado distribuido, generalmente solo usan un mecanismo de comunicación: Invocación Implícita o Sockets. Es por ello, que en este artículo se presenta una arquitectura flexible que permite realizar de forma distribuida el renderizado (proceso para generar una imagen 2D a partir de objetos 3D), el cual exige un alto consumo de recursos de procesamiento. La arquitectura permite configurar de manera dinámica tanto la técnica de renderizado como el mecanismo de comunicación.

Para realizar el renderizado distribuido de una escena 3D se deben revisar conceptos relevantes sobre sistemas distribuidos, computación gráfica y las técnicas de renderizado distribuido existentes, con el fin de determinar los requerimientos mínimos de la arquitectura de REDAG3D. Para probar dicha arquitectura, se debe implementar una aplicación prototipo que ponga en funcionamiento cada una de las características de los componentes definidos. Adicionalmente crear una escena constituida por modelos 3D complejos, que permitan validar las características de desempeño y flexibilidad de la arquitectura. Por otro lado, la aplicación debe tener un diseño consistente que ofrezca ventajas para la implementación, mantenimiento y puesta en marcha.

Las técnicas de renderizado distribuido fueron Sort-First y Sort-Last. Sus principales diferencias radican en la forma como se distribuye la geometría y en qué etapa del proceso de renderizado se hace el ordenamiento (sort). Sort-First implica redistribuir primitivas de espacio de pantalla a cada procesador, en donde se hace una clasificación de éstas antes de la etapa de transformación geométrica. En Sort-Last, las primitivas se envían a cada procesador sin una previa clasificación y se hace un ordenamiento de píxeles en la etapa de composición de la imagen final. Como mecanismos para la distribución se seleccionaron Sockets y JNDI-RMI. Sockets también fue utilizado como mecanismo de comunicación para el envío y recepción de mensajes entre las máquinas, y JNDI-RMI permitió la publicación de servicios para ser usados por la máquina que administra la aplicación. La elección de Sockets para el envío y recepción de mensajes se basó en unas pruebas cuyos resultados fueron analizados y arrojaron como conclusión que era el mejor mecanismo para esta labor.

Por otra parte, cuando se hace referencia al renderizado distribuido es necesario contar con escenas 3D que justifiquen la presencia de varias máquinas para realizar dicho proceso, por lo cual, se decidió crear una escena lo suficientemente compleja. Esta complejidad se mide según el número de mallas que conforman los modelos 3D de la escena, el número de triángulos de las mallas, las texturas asociadas a los modelos y efectos adicionales.

Se definió un plan de pruebas, el cual evalúa los aspectos más relevantes asociados al comportamiento de las técnicas de renderizado, los mecanismos de comunicación, así como el rendimiento y la funcionalidad de los gestores de distribución en la aplicación REDAG3D. La fase final corresponde al análisis de los resultados obtenidos de las pruebas, para posteriormente definir las conclusiones.

2. ANTECEDENTES

En esta sección se describen conceptos importantes tales como: los mecanismos de comunicación Sockets y JNDI, la arquitectura de renderizado de gráficos 3D, y las técnicas de renderizado distribuido Sort-First y Sort-Last.

Mecanismos de Comunicación
Los sockets proveen una comunicación full-duplex, punto a punto entre dos procesos. Son versátiles y son un componente básico de comunicación interprocesos e intersistemas. Un socket es un punto final de comunicación al cual se puede asociar un nombre, un tipo y uno o más procesos [1]. Por otra parte, JNDI es un API que proporciona funcionalidades de nombrado y directorio a las aplicaciones, y está definido para ser independiente de cualquier implementación de servicio de directorio. Las aplicaciones usan JNDI para acceder a una gran variedad de servicios de nombres y directorios. Se debe definir un SPI (Service Provider Interface) para hacer posible la conexión entre varios servicios de nombres y directorios, y así permitir que las aplicaciones accedan a los servicios de JNDI [2].

Arquitectura de Filtros y Tubos para gráficos 3D
Una arquitectura de tubos y filtros (pipeline) de renderizado de gráficos 3D convierte la representación geométrica de un mundo virtual 3D en una imagen 2D foto-realista [3], la figura 1 muestra las diferentes etapas del proceso. La entrada al pipeline es una escena conformada por un conjunto de objetos típicamente representados como mallas de triángulos. Las dos etapas principales son la transformación geométrica y la rasterización. En la etapa de transformación geométrica se mapean triángulos de un sistema de coordenadas 3D (espacio de objeto) a un sistema de coordenadas 2D (espacio de imagen) llevando a cabo una serie de transformaciones. En la etapa de rasterización se convierte los triángulos transformados en píxeles [4] [7] [10].

Figura 1. Pipeline de gráficos 3D

Algoritmo de Renderizado - Raytracing
Es un algoritmo computacional que permite generar imágenes "realistas". Se basa en la trayectoria seguida por rayos de luz individuales trazados desde la posición de la cámara hacia la escena. Estos rayos se intersectan con los objetos 3D de manera que se determinan las superficies visibles al observador [5] [8] [9] [11].

Renderizado Distribuido
Sort-First y Sort-Last son técnicas de renderizado distribuido cuya diferencia radica en la forma como se distribuye la geometría y en la etapa del proceso de renderizado en la cual se realiza el ordenamiento. En Sort-First, el espacio de pantalla se divide por regiones, las primitivas que pertenecen a cada región se clasifican según su ubicación, a cada procesador se le asigna una región y éste se encarga de renderizar las primitivas respectivas. Luego se realiza la transformación geométrica y la rasterización [6] [12]. En la Figura 2 se pueden ver las etapas del proceso de renderizado en la arquitectura Sort-First.

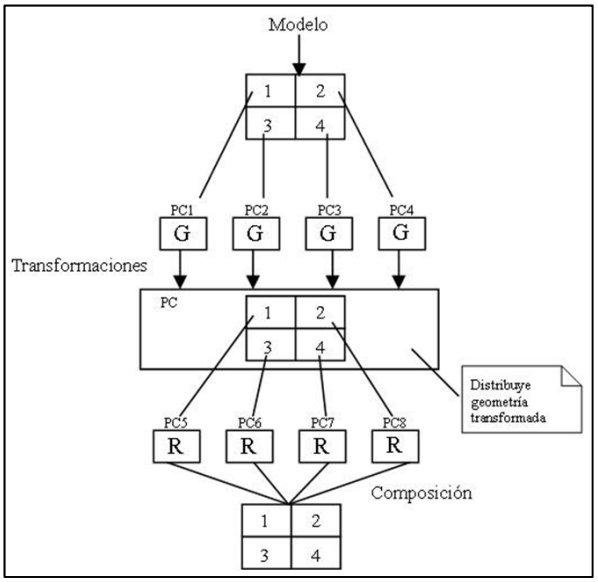

Figura 2. Etapas Sort-First

En Sort-Last cada procesador realiza el proceso de transformación geométrica y rasterización de las primitivas asignadas, posteriormente se lleva a cabo un proceso de composición de cada representación para formar la imagen final [6] [12]. En la Figura 3 se pueden ver las etapas del proceso de renderizado en la arquitectura Sort-Last.

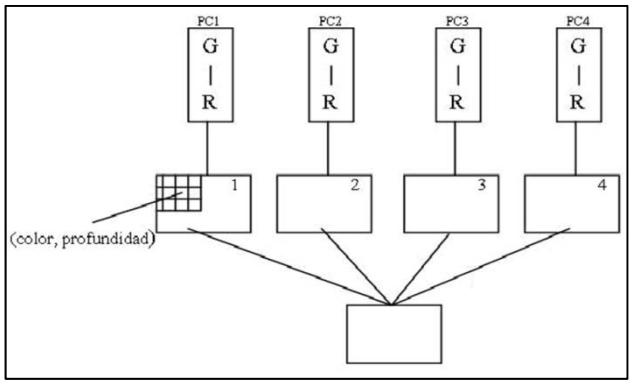

Figura 3. Etapas Sort-Last

3. DISEÑO E IMPLEMENTACIÓN DE REDAG3D

La arquitectura propuesta para REDAG3D, se definió de acuerdo a los requerimientos del sistema. En la Figura 4 se especifica la forma en que las máquinas están organizadas físicamente y el mecanismo de comunicación entre ellas (vista de despliegue). Existe un servidor encargado de la administración del sistema y varios clientes que pueden realizar solicitudes al mismo (máquinas visualizadoras). Sin embargo, el servidor actúa como cliente durante gran parte del proceso de renderizado, ya que utiliza los servicios prestados por las máquinas visualizadoras que se convierten en servidores de renderizado.

Se definió un estilo arquitectónico por capas, en donde cada capa ofrece servicios a las capas superiores. El cliente cuenta con una interfaz gráfica que permite seguir visualmente paso a paso el proceso que realiza la aplicación para renderizar una escena. Al inicio, el cliente establece la comunicación con el

servidor administrador a través del mecanismo Sockets, y este último mediante el mismo mecanismo establece conexiones con las máquinas visualizadoras. A través de estas conexiones, el servidor envía solicitudes a cada máquina para configurar la técnica de renderizado que será utilizada en el proceso y posteriormente el mecanismo de comunicación (Sockets/JNDI). Si se selecciona el mecanismo Sockets, se mantiene la comunicación que se estableció al inicio; si se selecciona JNDI, el servidor envía una solicitud de activación para que cada máquina visualizadora publique los servicios disponibles ante un Directorio de Nombres y Servicios (DNS). A partir de ese momento las máquinas se convierten en servidores y el servidor administrador consumirá sus servicios. Después de publicados los servicios, el servidor recupera el stub del servicio que va a solicitar y delega a cada máquina la carga correspondiente.

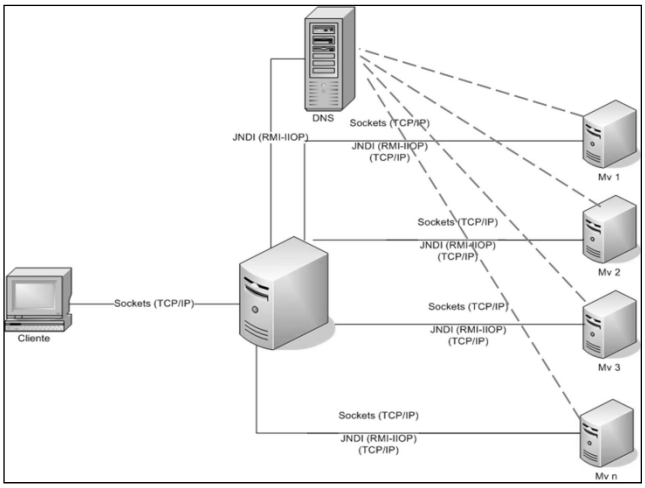

Figura 4. Arquitectura Cliente / Servidor

Debido a que el sistema es distribuido es necesario controlar la entrada o salida de una máquina visualizadora, para así mantener actualizado el número de servidores que están prestando el servicio de renderizado, esto debe ser verificado continuamente por el servidor administrador, y en caso de detectar la desconexión de una máquina, termina la comunicación con la misma.

Si se selecciona la técnica Sort-Last, las mallas se distribuyen a cada máquina visualizadora de manera aleatoria; si se selecciona la técnica Sort-First, cada malla se distribuye en las máquinas visualizadoras según una previa clasificación, dependiendo de su ubicación en el espacio de pantalla. Una vez se ha distribuido la carga, el servidor la envía a cada máquina a través del mecanismo de comunicación seleccionado. En cada máquina visualizadora se administra la geometría directamente, manipulando las mallas por medio de funciones que ofrece el toolkit VSDK (Vitral Software Development Kit), y posteriormente se renderiza la escena de acuerdo a la técnica seleccionada, retornando la imagen final al servidor administrador. Con Sort-Last el resultado del proceso de renderizado por cada máquina es una imagen y un buffer de profundidad, mientras que con Sort-First el resultado del proceso de renderizado por cada máquina es una sección de la imagen final. En las figuras 5 y 6 se puede ver un ejemplo (con 4 máquinas visualizadoras) de las imágenes resultado por cada máquina con las técnicas Sort-Last y Sort-First respectivamente.

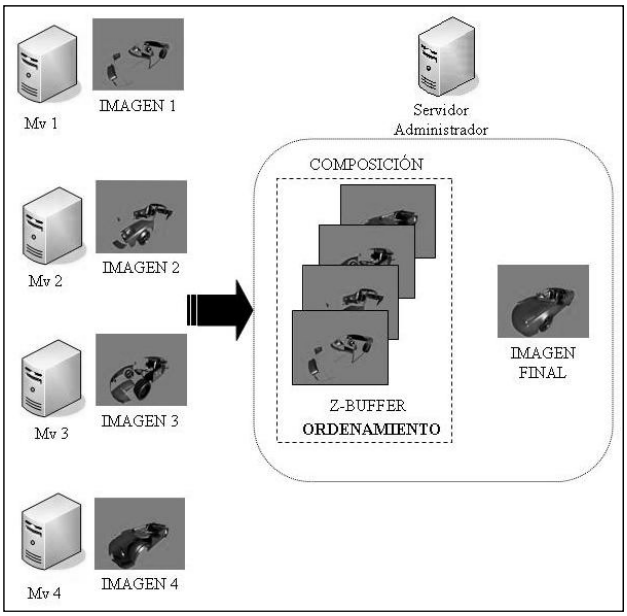

Figura 5. Composición en REDAG3D con Sort-Last (4 máquinas visualizadoras)

Una vez terminada la fase de renderizado el servidor administrador debe componer la imagen resultado para que sea presentada al cliente. Con Sort-Last se crea una imagen final aplicando el algoritmo de z-buffer. Con Sort-First se construye la imagen final uniendo las imágenes resultado. En las figuras 5 y 6 se puede ver la imagen final en el servidor después de aplicar las técnicas Sort-Last y Sort-First respectivamente.

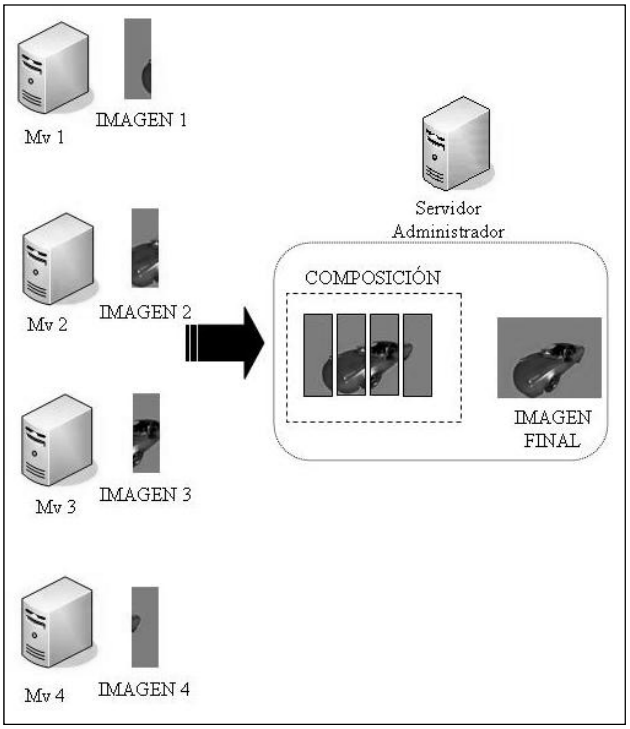

Figura 6. Composición en REDAG3D con Sort-First (4 máquinas visualizadoras)

Debido a los cambios en la información del sistema, el servidor tiene observadores que continuamente actualizan las interfaces cada vez que sucede un evento en la comunicación. Durante

todo el proceso se controlan los tiempos de respuesta al utilizar cada técnica de renderizado distribuido y cada mecanismo de comunicación.

4. PRUEBAS Y RESULTADOS

Con el fin de probar la arquitectura con los diferentes mecanismos de comunicación y técnicas de renderizado se especificaron pruebas de rendimiento y pruebas de renderizado. En todas las pruebas se utilizó la misma escena 3D, las variables son el número de máquinas, los mecanismos de comunicación y las técnicas de renderizado distribuido.

Pruebas de Rendimiento

Un punto crítico en los sistemas distribuidos es el mecanismo de comunicación, por tal motivo las pruebas de rendimiento comparan los mecanismos Sockets y JNDI, realizando escalamiento horizontal de las máquinas visualizadoras. En este tipo de prueba se registró el tiempo al iniciar la distribución de la carga en el servidor administrador, hasta finalizar la composición de la imagen resultado. En las Tablas 1 y 2 se muestran los resultados de estas pruebas, representados en las figuras respectivas.

Pruebas Rendimiento	
JNDI - Sort Last	
# Máquinas	Tiempo total (ms)
2	3177214
4	1715364,7
8	1208963,33
16	1101349
24	1042172

Pruebas Rendimiento	
Sockets - Sort Last	
# Máquinas	Tiempo total (ms)
2	2754526,3
4	1577572,7
8	1256884,3
16	1446161,3
24	1419193

Tabla No. 1: Pruebas Rendimiento Sort- Last

En la Figura 7 se observa que con 2 y 4 máquinas se presenta un mejor rendimiento al utilizar el mecanismo de comunicación Sockets, pero a partir de 8 máquinas en adelante mejora considerablemente el desempeño para JNDI-RMI.

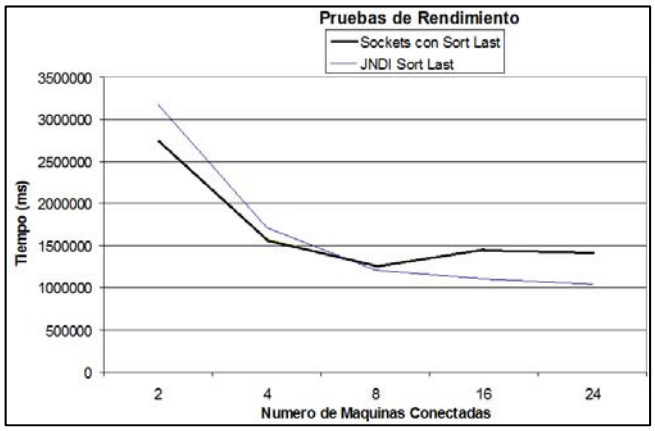

Figura 7.Sort-Last usando JNDI y Sockets

Pruebas Rendimiento	
JNDI - Sort First	
# Máquinas	Tiempo total (ms)
2	3130583
4	2356953,3
8	1382703
16	974635,3
24	867333,3

Pruebas Rendimiento	
Sockets - Sort First	
# Máquinas	Tiempo total (ms)
2	2607593,3
4	2369396
8	1943406,7
16	2256000
24	2321728,3

Tabla No. 2: Pruebas Rendimiento Sort-First

En la Figura 8 se observa al inicio un mejor rendimiento con Sockets, pero a partir de 4 máquinas el rendimiento cambia a favor de JNDI-RMI.

Por las pruebas de rendimiento se puede concluir que JNDI permite que el sistema sea escalable en cuanto al número de máquinas visualizadoras conectadas. Este mecanismo al no tener que hacer una interpretación de los datos que llegan (invocación directa) tiene un menor impacto en la comunicación comparado con Sockets.

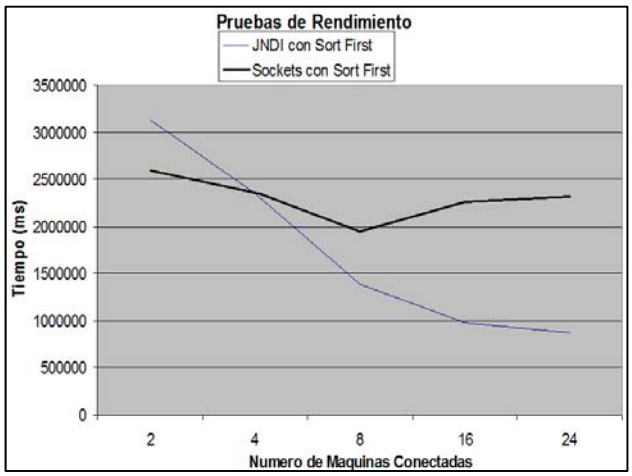

Figura 8. Sort-First usando JNDI y Sockets

Pruebas de Renderizado

Una vez seleccionado el mecanismo JNDI, como el que ofrece un mejor rendimiento a medida que aumenta el número de máquinas visualizadoras, se ejecutaron las pruebas de las técnicas de renderizado distribuido Sort-First y Sort-Last.

El propósito de las pruebas de renderizado es verificar la eficiencia de cada técnica de renderizado con el mismo mecanismo de comunicación. Para cada máquina renderizadora se tomó el tiempo de inicio y final del renderizado de las mallas asignadas, más el tiempo de envío de cada imagen al servidor administrador y el tiempo de composición de la imagen final. En el Figura 9 se muestran los resultados de estas pruebas.

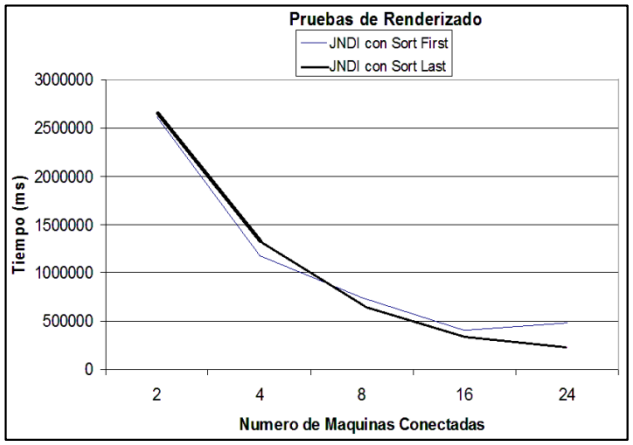

Figura 9. Sort-Last y Sort-First usando JNDI

Como se observa en la Figura 9, el comportamiento en general de las técnicas de renderizado es similar. Con 2 y 4 máquinas visualizadoras Sort-First es un poco más eficiente; a partir de 8 máquinas mejora el desempeño de Sort-Last. Con pocas máquinas Sort-First tiene un mejor comportamiento debido a que no se generan imágenes completas, por lo cual el envío de éstas a través de la red tarda menos tiempo que con Sort-Last. Cuando el número de máquinas es mayor a 4, el espacio de pantalla está subdividido en un gran número de regiones con Sort-First, por lo que se renderizan más de una vez mallas que ocupan varias regiones (ver Figura 10), debido a esto se observa un mejor comportamiento con Sort-Last.

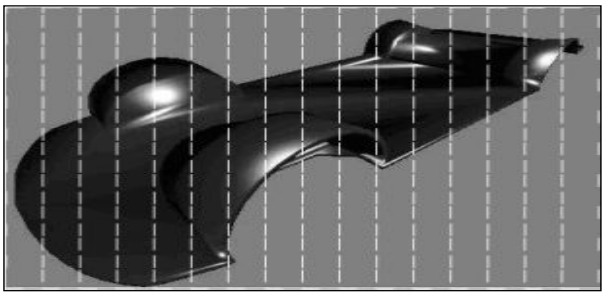
Figura 10. Malla que ocupa gran parte del viewport (16 regiones)

También se tuvieron en cuenta tiempos por cada máquina renderizadora de manera que se pudiera observar la distribución de la carga. En las Figuras 11 y 12 respectivamente, se pueden observar los resultados de estas pruebas para 24 máquinas con las dos técnicas de renderizado distribuido.

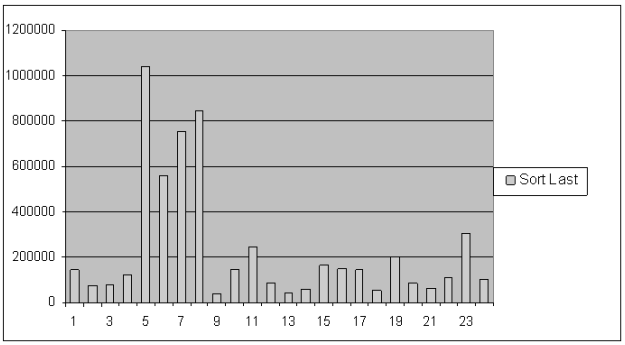

Figura 11. Sort-Last con JNDI (24 máquinas)

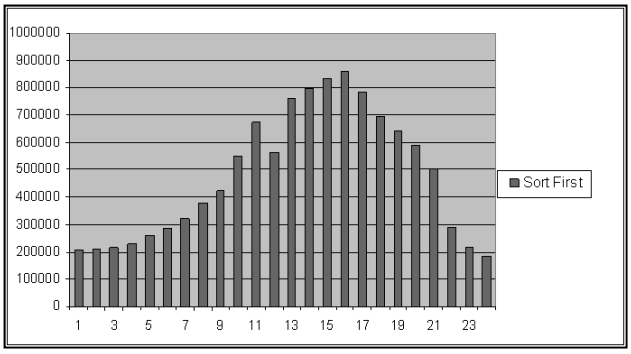

Figura 12. Sort-First con JNDI (24 máquinas)

Al observar los dos gráficos se puede concluir que la distribución es más uniforme con Sort-First. A las máquinas que demandaron más tiempo de renderizando les fueron asignadas una cantidad mayor de mallas, o mallas con un mayor nivel de complejidad respecto al número de triángulos.

5. CONCLUSIONES

La arquitectura definida para REDAG3D funciona correctamente y es escalable, presentando un rendimiento apropiado a medida que aumenta el número de máquinas visualizadoras. REDAG3D es portable gracias a la utilización del lenguaje JAVA y a que se pueden configurar los mecanismos de comunicación dependiendo del entorno; funcional, debido a que el usuario puede seleccionar la combinación Mecanismo de Comunicación/Técnica de Renderizado Distribuido que desee; integrable, puesto que puede ser fácilmente integrada a otra arquitectura, en este caso a Vitral, la cual es una arquitectura del grupo de computación gráfica de la Pontificia Universidad Javeriana, de igual forma los módulos implementados se pueden integrar al toolkit VSDK, desarrollado por el mismo grupo.

Una de las ventajas más notorias de acuerdo con los resultados obtenidos, es que JNDI-RMI al ser un mecanismo utilizado en el lenguaje JAVA, no necesita ser interpretado y realiza una invocación directa a los servicios que se necesitan consumir, por lo tanto reduce el tiempo de respuesta durante el proceso. Con el mecanismo de comunicación Sockets, los objetos que son enviados a través de la red se manejan a bajo nivel, mientras que con JNDI ocurre lo contrario, ya que permite la serialización de los mismos.

6. REFERENCIAS

[1] Los sockets en Java, Url: http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/V_2.htm, Última consulta: 18 de Mayo de 2008.

[2] JNDI Specifications for J2SE 1.5.0, Url: http://java.sun.com/products/jndi/reference/docs/index.html, Sun Developer Network, Última consulta: 18 de Mayo de 2008.

[3] I. Antochi, B. Juurlink, S. Vassiliadis. A Flexible Simulator for Exploring Hardware Rasterizers. Computer Engineering Laboratory, Delft University of Technology. 2002.

[4] T. Mitra, T. Chiueh. Compression-Domain Parallel Rendering. Proceedings of the 16th International

Symposium on Parallel and Distributed Processing, IEEE Computer Society. 2002.
[5] J.Foley, A. VanDam, S. Feiner, J. Hughes. “Computer Graphics. Principles and practice.” Addison-Wesley Publishing Company, Inc. 1991.
[6] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs. “A Sorting Classification of Parallel Rendering”. IEEE Computer Graphics and Applications. Volume 14, July 1994 pp(s):23-32 1994.
[7] O. Staadt, J. Walker, C. Nuber, B. Hamann. “A Survey and Performance Analysis of Software Platforms for Interactive Cluster-Based Multi-Screen Rendering”. The Eurographics Association. 2003.
[8] R. Samanta, T. Funkhouser, K. Li, and J. Pal Singh, Hybrid Sort-First and Sort-Last, Parallel Rendering with a Cluster of PCs, In Eurographics/SIGGRAPH workshop on Graphics, pp 99--108. ACM Press, 2000
[9] Carl Mueller, The Sort-First Rendering Architecture for High-Performance Graphics, Computer Graphics, ACM SIGGRAPH Special Issue on 1995 Symposium on Interactive 3-D Graphics, April 1995.
[10] Watt Alan, 3D Computer Graphics, Addison Wesley, Tercera Edición, 2000.
[11] Tatarchuk Natalya, Advanced Real-Time Rendering in 3D Graphics and Games, SIGGRAPH 2007, Course 28, August 8, 2007.
[12] Krishna P.C. Madhavan, Laura L. Arns, and Gary R. Bertoline, A Distributed Rendering Environment for Teaching Animation and Scientific Visualization, IEEE Computer Society, September/October 2005.