MULTICIENCIAS, Vol. 14, Nº 1, 2014 (88 - 96)
ISSN 1317-2255 / Dep. legal pp. 200002FA828

Diseño de clústeres computarizados con algoritmos de renderizado para la construcción de modelos tridimensionales a través del uso de herramientas *open source*

Ulises Olivares Pinto[1], Heberto Ferreira Medina[2], Carlos Arturo Vega Lebrún[3], José Luis Cendejas Valdez[4] y Genoveva Rosano Ortega[3]

[1,4]*Instituto Tecnológico de Morelia. México.*
[2]*Centro de Investigaciones en Ecosistemas (CIEco), UNAM. México.*
[3]*Universidad Popular Autónoma del Estado de Puebla, Campus Central. México.*

uolivares@gdl.cinvestav.mx; hferreir@cieco.unam.mx; genoveva.rosano@upaep.mx; luiscendejas@hotmail.com

Resumen

Este artículo propone la generación de mejores prácticas relacionadas con la construcción de clusters que permitan recorridos tridimensionales (3D) de calidad con una considerable reducción en el tiempo de ejecución del renderizado. En la actualidad el uso de software y de tecnología que reduzca el tiempo de procesamiento en el renderizado de recorridos 3D es importante ya que realizarlo en una computadora personal consume gran tiempo de procesamiento. La demanda en la construcción de recorridos virtuales 3D, la ejecución y su complejidad hace necesario realizar este proceso a través de clústeres de computadores. Además el análisis de las herramientas y técnicas más utilizadas, donde la velocidad, la complejidad y el conocimiento son de vital importancia para generar recorridos en forma rápida. El objetivo de este trabajo es optimizar el tiempo de renderizado de un recorrido virtual 3D, utilizando un clúster de computadoras. Se utilizó cómputo paralelo además de herramientas basadas en software libre como: 1) OpenMosix, que permite la construcción de un clúster para paralelizar la carga de trabajo en el proceso de renderizado y 2) Blender para diseñar modelos 3D; la compatibilidad de las dos herramientas se basa en ser bajo licencia GPL (software libre).

Palabras clave: recorridos 3D, computación en paralelo, clúster, software libre.

Recibido: 22-04-2013 / Aceptado: 13-11-2013

Design of Computer Clusters with Rendering Algorithms for Constructing 3d Models Using Open Source Tools

Abstract

This paper proposes the creation of best practices related to constructing clusters based on OpenMosix, which permits developing quality, three-dimensional virtual tours with a considerable reduction in time for executing the rendering. Nowadays, is very important to use software and technology capable of reducing rendering process time for three-dimensional tours, because doing this on a personal computer can take too long. Demand for the construction of virtual 3D tours, its execution and complexity make it necessary to carry out the process using a cluster of computers. It is also vital to analyze the tools and techniques most used today for rapid 3D tour generation. The objective of this work is to optimize rendering time for three-dimensional virtual tours using a cluster of computers based on free software. Parallel computing was used in addition to free software tools, such as: 1) OpenMosix, which permits constructing a cluster to parallelize the workload in the rendering process and 2) Blender, for rendering 3D models. Compatibility between the tools is optimal because they have GPL licenses (free software).

Keywords: virtual tours, parallel computing, cluster, free software.

Introducción

La investigación en sistemas distribuidos y paralelos es una de las líneas de mayor desarrollo en la ciencia informática en las últimas décadas (Grama, Gupta, Karypis & Kumar, 2003), (Dongarra *et al*. 2003). Además de la utilización de elementos virtuales que asemejan la realidad a través de un equipo de cómputo (3D) se ha vuelto común en las actividades de la vida diaria, en donde debido al gran número de tareas que se tienen que procesar, se vuelve necesario contar con equipos de cómputo que cuenten con mayores capacidades técnicas, para la incorporación de nuevas tecnologías, además de realizar este trabajo sin tener que sacrificar aspectos como la velocidad de procesamiento, el tiempo de ejecución y la calidad en los resultados. Según (Morcillo & Vallejo, 2009), una posible solución a este problema de cómputo es utilizar algoritmos y mecanismos de validación que contribuyan al mejoramiento del renderizado. Por su parte (Cebolla, 2006), comenta que este proceso es el mecanismo para obtener una imagen o un video a partir de la descripción de una escena 3D. Visualización es el proceso de representar datos científicos abstractos con imágenes 3D que ayudan a entender el significado de estos datos (Sánchez, 2004).

El renderizado de recorridos virtuales es un proceso muy complejo y demandante, el cual requiere de un gran poder de procesamiento computacional, por lo tanto entre más componentes gráficos (texturas, cámaras, objetos) estén integrados en un proceso único, el tiempo de procesamiento se incrementa sustancialmente llegando a un punto en el que el tiempo es un factor que define el éxito de un proyecto de animación, por lo que la necesidad de tener mayor poder de procesamiento para disminuir el tiempo de renderizado se hace cada vez más necesario. Las ventajas de los sistemas de archivos distribuidos respecto a los sistemas centralizados son según (Schroeder, 1993): 1) Compartir recursos: los usuarios tienen acceso a sus recursos compartidos a través de la red; 2) Reducción de costos: las redes de equipos de cómputo ofrecen una mejor relación precio/rendimiento que los mainframes; 3) Tolerancia a fallos: los sistemas distribuidos replican recursos y procesos, a fin de proporcionar fiabilidad y disponibilidad al sistema. Una solución a este problema, es la implementación del cómputo paralelo a través de un clúster y/o grid de computadoras bajo OpenMosix; es un software que paraleliza procesos en GNU/Linux, migrando dichos procesos de una forma dinámica. Consiste en algunos algoritmos que comparten recursos a nivel de kernel y que están enfo-

cados a conseguir un alto rendimiento, escalabilidad con baja sobrecarga y un clúster fácil de utilizar.

La idea esencial es que los procesos colaboren de forma que simulen que están en un mismo nodo. Los algoritmos de OpenMosix son dinámicos ofreciendo una gran ventaja frente a los algoritmos estáticos de PVM/MPI, estos responden a las variaciones de los recursos entre los nodos migrando procesos de un nodo a otro de forma transparente para el proceso, para balancear la carga y evitar la falta de memoria de un nodo (Moshe, 2004). Desde 1955 se han investigado las arquitecturas paralelas obteniendo parámetros para optimizarlas, así como la obtención de la relación costo-rendimiento. El paradigma de computación paralela se ha aplicado a diversas ramas de la ciencia y ha cobrado avances significativos en distintas disciplinas y áreas hasta el punto en que no sólo universidades e instituciones dedicadas a la investigación científica requieren de capacidades y poder de cómputo que brinda un clúster. La iniciativa privada ha incursionado en el ámbito del cómputo paralelo y grandes proyectos han surgido de la aplicación de este paradigma, por citar algunos proyectos que se han llevado a cabo en la iniciativa pública, privada y en centros de investigación (Herrera, Carrillo & Hernández, 2007). Los sistemas distribuidos han tenido un gran auge y han sido importantes en ámbitos como: medicina, animación, cinematografía, arquitectura, física, matemáticas, química, entre otras, ya que se destacan como sistemas escalables por excelencia. Al poseer una escala se tiene una relación directa con tres factores importantes: 1) costo-rendimiento; 2) número de equipos-rendimiento; 3) compatibilidad-rendimiento. El procesamiento paralelo se entiende como un término que se utiliza para denotar un grupo de técnicas significativas que se usan para proporcionar tareas simultáneas de procesamiento concurrente de datos para conseguir un menor tiempo de ejecución (Morris, 1994). El procesamiento paralelo se basa en dividir un problema en un conjunto de partes resolubles de forma concurrente, de manera que el tiempo total de solución se divide por el número de procesadores utilizados (Pardines, 2005). Hoy día, los clústeres computacionales son considerados como una plataforma estándar para la computación de alto desempeño (*high performance computing)* (Dongarra, 2011).

Según (Cendejas, Vega, Ordoñez & Ferreira, 2012), para realizar el desarrollo de un recorrido virtual, es necesario contar con un levantamiento de los diferentes requerimientos que contendrá: 1) Dimensión, 2) Recorrido de la animación, 3) Efectos, 4) Tiempo, 5) Iluminación del escenario, 6) Cámaras y 7) Volcado (NTSC/PAL/SECAM). Muchos de los escenarios de simulación 3D dedicados a la realidad virtual utilizan el modelo geométrico para su diseño (Lozano & Calderon, 2004).

Blender permite el modelado, animación y creación de gráficos tridimensionales, una característica importante es que es multiplataforma en la actualidad por su compatibilidad con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX. En Blender es posible el renderizado con el motor que utiliza por defecto, adicionalmente se controla y automatiza tareas mediante scripts de python (Mullen, 2009). Blender cuenta con un motor de render denominado Yaf-Ray, que se incluye de forma predeterminada, sin embargo, al igual que en otros software de modelado en 3D como 3ds Max y Maya, se pueden utilizar motores de renderizado extras que proveen un conjunto de texturas y librerías que generan modelos más reales tales como: Pov-Ray o Blental (Versión de Mental-Ray para Blender). De ahí la importancia de diseñar en Blender y llevar a cabo sus renderizados a través de un clúster construido sobre OpenMosix. Las ventajas que presenta el software libre en la actualidad se ven reflejadas en la Tabla 1.

Metodología

El proyecto se realizó en diferentes etapas las cuales incluyen el proceso de diseño del recorrido 3D, agregar la biblioteca con los objetos requeridos, aplicar el motor de renderizado y finalmente la ejecución en el clúster, como se muestra en la Figura 1.

La creación de un modelo 3D sigue una serie de pasos, que según (Moreno, 2010), son los siguientes: 1) modelar el objeto por medio del uso de polígonos y otros elementos geométricos; 2) se pasa a las etapas de texturizado e iluminación, que son propiedades visuales del comportamiento del objeto frente a la luz, fundamentales para que el gráfico sea llamado modelo 3D; y 3) la animación y renderizado del objeto en una escena. Una vez construido el modelo 3D, se llevan a cabo las etapas del proceso en el cuál se construye cada fotograma para crear una animación o película en 3D, generando el recorrido virtual, es necesario que el programa que se use para modelar, construya una foto de la cámara virtual que está visualizando el escenario y así pasar a la etapa de renderizado de los frames de la línea de tiempo, para obtener el volcado en cualquier formato de video.

Para el renderizado de animaciones de modelos 3D existen varias formas para abordar este proceso en una animación que demande muchos recursos, una de ellas es la solución tradicional del problema, es decir realizar el proceso de generación de una animación en una sola com-

Tabla 1. Ventajas del software en el desarrollo de modelos 3D.

Indicador	Software libre	Software propietario
Costo de inversión menor	✓	
Libertad de uso y redistribución	✓	
Independencia tecnológica	✓	✓
Costo por licencia		✓
Formatos de archivo único	✓	✓
Seguridad en el sistema		✓
Eficiencia en control de fallos		✓

Fuente: propia.

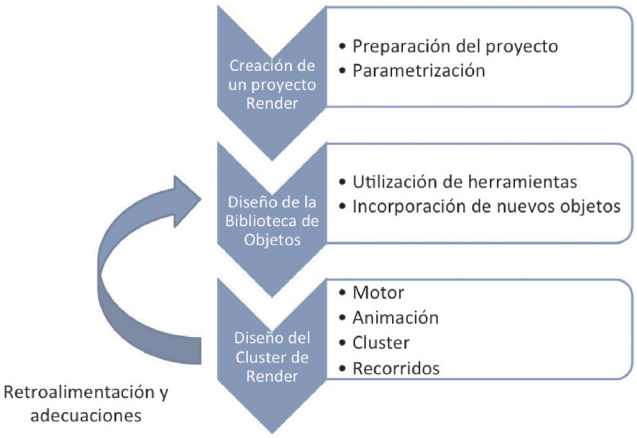

Figura 1. Etapas para el desarrollo de un modelo 3D utilizando un clúster de computadoras. Fuente: propia.

putadora haciendo uso de técnicas tradicionales de ejecución, al tener sólo un equipo de cómputo que realiza la tarea de renderizado se obtienen tiempos de espera demasiado altos. Es aquí donde el paradigma de cómputo paralelo cobra una gran importancia puesto que al adoptar esta propuesta metodológica se divide en distintas tareas para realizar este proceso, "divide y vencerás". De esta forma se observa una disminución considerable en el tiempo de renderizado, con lo que se impacta directamente en la premisa propuesta, en los objetivos planteados, en el marco conceptual de desarrollo de software, lo cual genera una solución viable para la solución del problema identificado.

Desarrollo

En la primer etapa se construyeron algunos modelos 3D para generar su proceso de renderizado, dichos modelos fueron creados en Blender, este formato permite la compatibilidad con software de modelado en 3D como: 3Ds Max, Maya, Cinema 4D, Lightwave 3D entre otros.

En la segunda etapa del proyecto, se procedió a la instalación de las aplicaciones que permiten el trabajo con el clúster. Según (Moshe, 2004); menciona que un sistema distribuido es un conjunto de ordenadores o procesadores independientes en donde cada usuario funciona como uno solo. Un clúster es un grupo de computadoras interconectadas que trabajan conjuntamente en la solución de un problema (Colobran H. & Arqués S, 2008). Para la construcción del clúster se instaló una distribución de Linux, Cluster-Knoppix es una distribución basada en Debian y que se puede descargar en formato live, proporcionando una combinación entre la accesibilidad que brinda Knoppix y la funcionalidad de OpenMosix. Esta distribución incluye el kernel con la mejora (parche) de OpenMosix, las herramientas de área de usuario y también posee la herramienta de OpenMosix View que ayuda al monitoreo y administración del clúster (Rankin, 2008). Otra ventaja de Cluster-Knoppix es que integra la terminal de servidor de OpenMosix denominado Pre-Boot Execution Enviroment (PXE), que permite realizar un arranque en red para evitar instalar el sistema operativo en cada nodo, por lo tanto el sistema de ficheros puede estar compartido con todos los nodos del clúster.

El administrador cuenta con una aplicación principal que permite interactuar y conocer las distintas herramientas además de conocer el ID del nodo que OpenMosix asigna a cada una de las computadoras de forma automática, este número es el que se usa para acceder a cada nodo dentro del clúster, indicando así un estado operativo cuando se encuentra en verde y no operativo cuando se encuentra en rojo. Se muestra una lista de las direcciones IP con las que se identifican los nodos para la transmisión de los paquetes, esta configuración de direcciones IP se puede realizar en forma automática mediante el protocolo DHCP. Esta misma interface permite visualizar la línea de tiempo que establece la velocidad que el kernel de OpenMosix considera para cada uno de los nodos del clúster, así los nodos con mayor velocidad recibirán mayor carga de trabajo. Las barras de progreso indican el porcentaje de eficiencia del algoritmo de balanceo de carga, la carga del sistema en general y el porcentaje de uso de memoria. De la misma forma se muestra la cantidad de memoria y procesadores que tiene cada nodo. Finalmente aparece un indicador del estado de OpenMosix Collector que se encarga de recolectar información del sistema y proporcionarla a los demonios de OpenMosix, como se muestra en la Figura 2.

Los nodos que constituyen el clúster son computadores PC integrados mediante una red de área local o una red de sistema (Pardines, 2005). La Figura 3 muestra cómo se realiza la división de bloques (frames) en el clúster y el proceso de render a través del flujo bidireccional; entre el

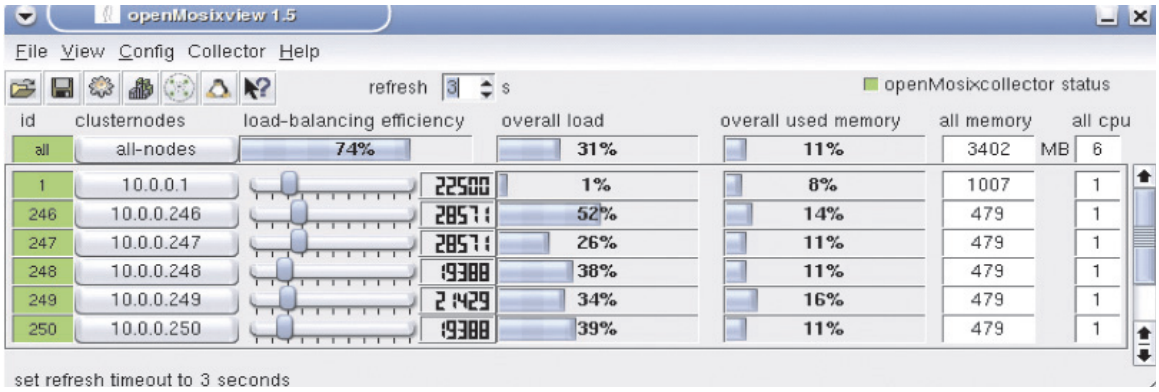

Figura 2. Administrador de OpenMosix. Fuente: propia.

nodo maestro y los nodos esclavos, al realizar el envío y la recepción de frames se genera un tráfico bidireccional; inicialmente se recibe la instrucción para realizar el renderizado de un frame y posteriormente se retorna el número de frames asignados a cada nodo hacia el nodo maestro, para integrar el proyecto animado. Las soluciones paralelas a este problema puede diferir en cuanto a la configuración de la comunicación que se establece, es decir en la frecuencia y cantidad de datos transferidos punto a punto entre procesadores (Choi, 1997).

Para la división de frames se utilizó un algoritmo y su programación en scripts de OpenMosix, los algoritmos se muestra en la Figura 4. A través de estos algoritmos se realiza la migración de procesos en OpenMosix. El algoritmo 2 divide una animación en Blender y realiza la división de trabajo entre los nodos del clúster, tomando como base de desarrollo el algoritmo propuesto por (Gloor, 2004).

Resultados

Para la obtención de resultados se realizaron pruebas de renderizado con diversos proyectos de animación en 3D, cada uno con distintos requerimientos computacionales. Se obtuvieron resultados con tres proyectos distintos de animación, en donde se realizó el cálculo del *Speed Up* con distinto número de procesadores y se obtuvo una disminución significativa en el tiempo de renderizado.

Proyecto 1. Consta de 250 frames, una vez renderizado se obtiene como producto final un video con una duración aproximada de 10 segundos y un tamaño total de 1.61 GB. En la Tabla 2 se muestran las características del modelo en donde los datos proporcionados muestran la información de tamaño inicial, tamaño final, dimensiones de los frames y tiempo de renderizado, por frame.

Al realizar el renderizado con diferente número de nodos cada uno con 2 cores se obtienen los resultados mostrados en la Tabla 3.

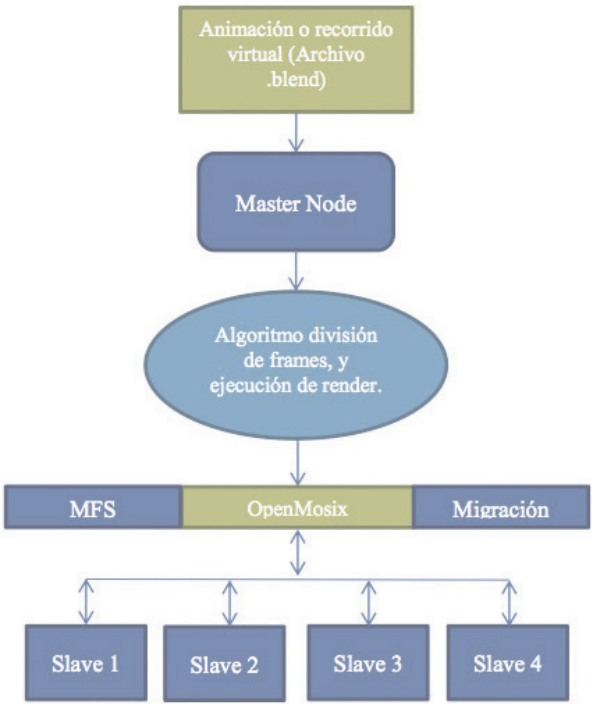

Figura 3. Procesamiento y división de frames en el clúster. Fuente: propia.

Proyecto 2. Este proyecto consta de 50 frames, una vez renderizado se obtiene como producto final un video con una duración aproximada de 3 segundos y un tamaño total de 554 MB. En la Tabla 4, se observan las características del modelo en donde los datos muestran la información de tamaño inicial, final, dimensiones y tiempo de renderizado por frame.

Al realizar el renderizado con diferente número de nodos cada uno con 2 cores se obtienen los resultados mostrados en la Tabla 5.

Proyecto 3. Este proyecto consta de 10 frames, una vez renderizado se obtuvo como producto final un video con una duración aproximada de 2 segundos y un tamaño total

```
   input  : Un conjunto de nodos I = {i1, i2, i3, ... in} n ∈ N.
   output: Un nodo (Master Node) con mejores recursos que los
           demás
 1 masterNode=-1;
 2 numberOfNodes=0;
 3 main()
 4 |  // I = {i1, i2, i3, ... in} n ∈ N Nodos del cluster
   |  numberOfNodes=|I|;
 5 |  masterNode=selectMasterNode(I);
 6 |  // Se envía la animación al nodo maestro
 7 |  sendDirectory(path, masterNode);
 8 end
 9 Function selectMasterNode(I)
10 |  numResources=-∞;
11 |  amountOfMemory=-∞;
12 |  processorSpeed=-∞;
13 |  masterNode=-1;
14 |  // Un nodo n ∈ N poseen un conjunto de recursos R(ik)
15 |  for n in I do
16 |  |  // m ∈ R(ik) tamaño de memoria de un nodo
17 |  |  // p ∈ R(ik) velocidad de procesador de un nodo
18 |  |  if |R(ik)| >= numResources and m ∈ R(ik) >
   |  |  amountOfMemory and p ∈ R(ik) > processorSpeed then
19 |  |  |  maxResources=|R(ik)|;
20 |  |  |  amountOfMemory=m ∈ R(ik);
21 |  |  |  processorSpeed=p ∈ R(ik);
22 |  |  |  masterNode=k;
23 |  |  end
24 |  end
25 |  // Se retorna un nodo con mejores recursos
26 |  return masterNode;
27 end
```

Algoritmo 1. Selección de un nodo maestro (un nodo con mejores recursos, si los nodos son iguales se selecciona el primero). Este script se ejecuta en cualquier nodo.

```
   input  : Una animación con extension .blend.
   output: Un conjunto de frames renderizados pertenecientes a la
           animación.
1 Function startRender(animation, numberOfNodes)
2 |  projectName=animation.name();
3 |  numberOfFrames=animation.end()-animation.start();
4 |  initialFrmae=animation.start();
5 |  finalFrame=animation.end();
6 |  // Se instancia script propuesto por (Gloor,2004)
7 |  render(projectName, initialFrame, finalFrame,numberOfNodes);
8 end
```

Algoritmo 2. Algoritmo que se ejecuta en el nodo maestro y ejecuta un ShellScript propuesto por Gloor para que a través de OpenMosix se realice la división de trabajo.

Figura 4. Algoritmo 1 y 2 para selección del nodo maestro y la división de frames. Fuente: propia.

de 301 MB. En la Tabla 6 se muestra las características del modelo en donde los datos proporcionados muestran la información de tamaño inicial, tamaño final, dimensiones de los frames y tiempo de render, por frame.

Al realizar el renderizado con diferente número de nodos cada uno con 2 cores, se obtienen los resultados mostrados en la Tabla 7.

Podemos observar la evaluación del *Speed Up* del proyecto 1 en donde se obtuvo una ganancia de tiempo significativa al llegar a disminuir el tiempo de renderizado en la pruebas realizadas para el modelo 1, con un mínimo de 167% con 2 nodos cada uno con 2 cores por nodo, es decir en un sistema con 4 cores. Y un máximo 740% con 10 nodos cada uno con 2 cores por nodo, es decir en un sistema con 20 cores, como se muestra en la Figura 5. Podemos observar la evaluación del *Speed Up* del proyecto 2 y 3 los cuales contaron con 2, 4, 6 y 10 nodos, donde se obtuvieron una línea de tendencia polinomial, como se muestra en la Figura 5.

En la Figura 6 se muestra el resultado de un frame de la animación generada una vez completado el proceso de renderizado del modelo tridimensional del proyecto 1, 2 y 3 respectivamente.

Discusiones y conclusiones

Mediante la construcción de un clúster basado en soluciones de software libre como: OpenMosix y Blender, es posible disminuir el tiempo de renderizado de recorridos virtuales o animaciones de modelos 3D, se logró a través de la programación de un script que permite la división de

Tabla 2. Características del proyecto 1 por frame.

Proyecto Blender	Tamaño inicial	Tamaño final	Dimensiones	T. render por frame
robot.blend	368 KB	6.6 MB	1024x768	8.73 seg

Fuente: propia.

Tabla 3. Resultado del renderizado del proyecto 1 con diferente número de nodos y cores.

No. Nodos	No. Cores	Tiempo (min)	Speed Up	Dism. tiempo
1	2	37.4	-	-
2	4	22.3	1.67	167%
4	8	11.16	3.35	335%
6	12	7.63	4.90	490%
10	20	5.05	7.40	740%

Fuente: propia.

Tabla 4. Características del proyecto 2 por frame.

Proyecto Blender	Tamaño inicial	Tamaño final	Dimensiones	T. render por frame
marble.blend	3.4 MB	11.08 MB	800x600	18.95 seg.

Fuente: propia.

Tabla 5. Resultado del renderizado del proyecto 2 con diferente número de nodos y cores.

No. Nodos	No. Cores	Tiempo (min)	Speed Up	Dism. tiempo
1	2	15.86	-	-
2	4	8.26	1.04	104%
4	8	7.45	2.12	212%
6	12	4.41	3.59	359%
10	20	1.95	8.13	813%

Fuente: propia.

Tabla 6. Características del proyecto 3 por frame.

Proyecto Blender	Tamaño inicial	Tamaño final	Dimensiones	T. render por frame
fire.blend	223 KB	30.10 MB	800x600	4.85 min.

Fuente: propia.

Tabla 7. Resultado del renderizado del proyecto 2 con diferente número de nodos y cores.

No. Nodos	No. Cores	Tiempo (min)	Speed Up	Dism. tiempo
1	2	15.86	-	-
2	4	8.26	1.04	104%
4	8	7.45	2.12	212%
6	12	4.41	3.59	359%
10	20	1.95	8.13	813%

Fuente: propia.

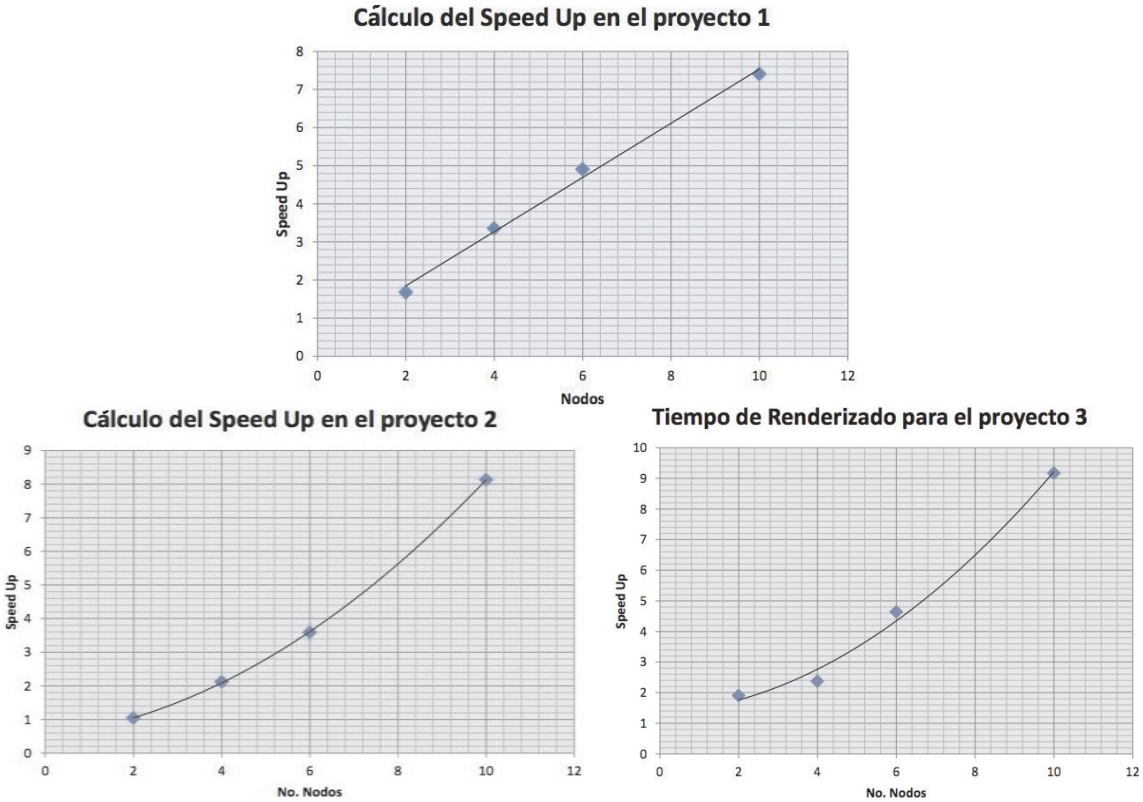

Figura 5. Cálculo de Speed Up del proyecto 1, 2 y 3. Fuente: propia.

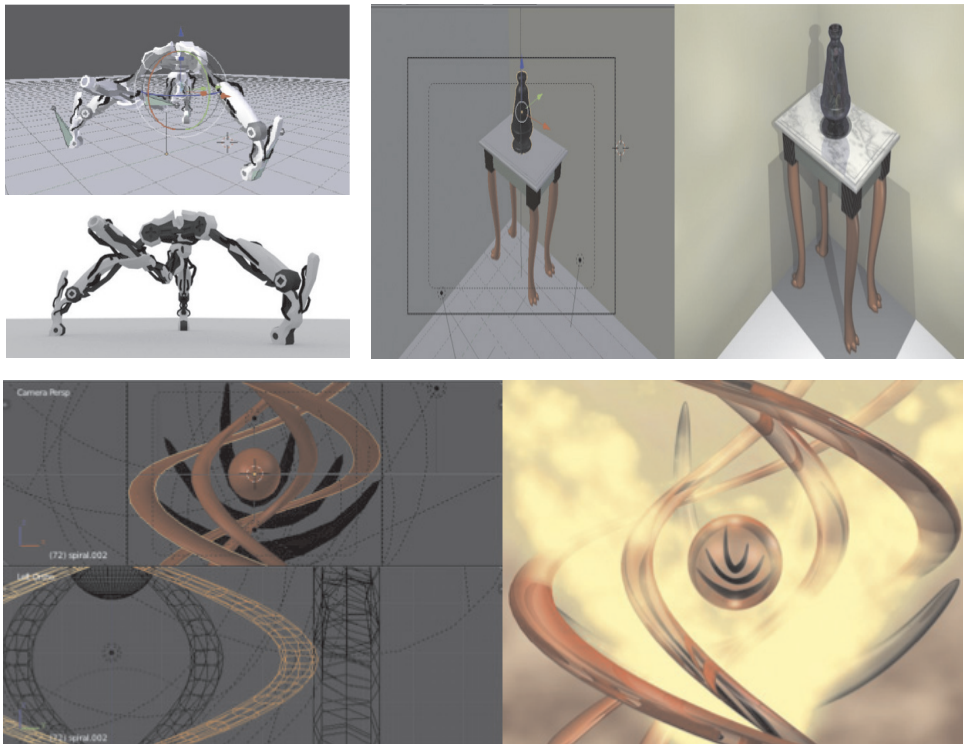

Figura 6. Resultado de la animación de los proyectos 1, 2 y 3. Fuente: propia.

carga de trabajo en el cluster. Dicho algoritmo muestra una mejora en el render de un proyecto 3D, ya que el *Speed Up* por el número de nodos fue mayor. Con esto se comprueba que con base en los resultados obtenidos, entre menos tráfico se genera entre el nodo maestro y los nodos esclavos más eficiente es el proceso de renderizado, puesto que de esta forma se evitan tiempos de espera innecesarios al enviar y recibir paquetes en el renderizado, por lo que la división de frames y la repartición de la carga de trabajo debe hacerse tomando en cuenta esta premisa.

A través de la experimentación se observó que el nodo maestro a través del cual se realiza la migración de los procesos, debe tener características en igualdad o superioridad respecto de los nodos esclavos (algoritmo 1), ya que la división de trabajo debe hacerse en tiempo real y de forma eficiente para evitar tiempos de espera y por consecuencia una disminución en el *Speed Up*. La velocidad de transmisión de información de la red de interconexión juega un papel vital en el desarrollo de un cluster, puesto se realiza la migración de los procesos y la transmisión de los frames renderizados hacia el nodo maestro (algoritmo 2). Por lo tanto, se concluye que el incremento en la velocidad de la red es significativamente mayor y mejora el *Speed Up*, cuando se tiene un gran número de nodos interconectados. Con base en la experimentación se observó que la virtualización de computadoras no arrojó resultados mejores en el proceso de renderizado, puesto que se dividen los recursos de un solo equipo entre las n máquinas virtuales y este proceso requiere poder de cómputo independiente.

Aunque en esta investigación se utilizó un ShellScript secuencial, la división de trabajo y la ejecución de tareas en forma simultánea permitió mejoras en el tiempo de obtención de las imágenes y el recorrido 3D por consecuencia.

La presente investigación permitió contemplar trabajos futuros en donde se implementen mejoras como:

- Mejorar la compatibilidad del OpenMosix con versiones de Linux GNU más actualizadas.
- Diseño de una interfaz gráfica que automatice el renderizado, para que un usuario estándar pueda generar una animación sin conocimientos sobre los comandos de Shellscript y de Linux.
- Desarrollar un clúster dedicado exclusivamente al renderizado y ejecución de código en paralelo para comercializar soluciones resueltas mediante cómputo paralelo.
- Implementar herramientas de OpenMosix que exploten la utilización de GPU's a través de CUDA y Open GL, para incrementar el rendimiento del clúster.
- Comprobar el funcionamiento del clúster con proyectos de mayor tamaño y más cantidad de escenas.
- Implementar las herramientas propuestas utilizando tecnologías de red más rápidas y eficientes.

Agradecimientos

Agradecemos al MTI. Hugo Zavala V. del CIGA-UNAM, al MTI. Alberto Valencia G. y a la Ing. Atzimba López M. del CIEco-UNAM, por su apoyo en la programación y ejecución de los algoritmos en el clúster.

Referencias

CEBOLLA, C. (2006). **3D studio Max.** Madrid, España: Alfaomega.

CENDEJAS V., J.L.; VEGA L., C.A.; TOLEDO O., O.; MEDINA F., H. (2012). Rendering of Virtual Tours of Three Dimensional Model for Application in Higher Education. IJCSNS **International journal of computer science and network security**, 12(6), 116-121.

CHOI, J. (1997). A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers, **Proceedings of the High-Performance Computing on the Information Superhighway,** IEEE, HPC-Asia.

COLOBRAN H., M.; ARQUÉS S., J.M. (2008). **Administración de Sistemas Operativos en Red.** Editorial UOC.

DONGARRA, J.; FOSTER, I.; FOX, G.; GROPP, W.; KENNEDY, K.; TORCZON, L.; WHITE, A (2003). **The Sourcebook of Parallel Computing.** Morgan Kauffman Publishers. Elsevier Science.

DONGARRA, J. (2011). Japan Reclaims Top Ranking on Latest TOP500 List of World's Supercomputers. Más reciente consulta: 17 de julio 2012. Disponible en el sitio: Top500 Supercomputer Site. http://www.top500.org

GRAMA, A.; GUPTA, A.; KARYPIS, G.; KUMAR, V. (2003). **Introduction to parallel computing.** Second Edition. Pearson Addison Wesley,

GLOOR O., Mak (2004). Blender load balancing workload manager. Recuperado de: http://spot.river-styx.com/downloads/mosix-blender.html Más reciente consulta: el 2 de Agosto del 2011.

HERRERA R, I.; CARRILLO L, A.; HERNÁNDEZ G., G. (2007). **Aplicación del cómputo paralelo a sistemas continuos.** [Versión Electrónica]. Instituto de geofísica de la Universidad Nacional Autónoma de México (UNAM). http://www.mmc.igeofcu.unam.mx/ Más reciente consulta: el 4 de Julio del 2011.

LOZANO, M.; CALDERÓN, C. (2004). Entornos virtuales 3D clásicos e inteligentes: hacia un nuevo marco de simulación para aplicaciones gráficas 3D interactivas. Inteligencia Artificial. **Revista Iberoamericana de Inteligencia Artificial.**

MOSHE, B.; BUYTAERT, K. (2004). The OpenMosix How To. http://openmosix.sourceforge.net/documentation.html Más reciente consulta: el 7 de Junio del 2011.

MORCILLO, C.; VALLEJO D. (2009)**. Fundamentos de Síntesis de Imagen 3D. Un Enfoque práctico a Blender.** Ebook, Escuela Superior de Informática y el Centro de Excelencia del Software, Universidad de Castilla-La Mancha.

MORENO S., J.; MOLINA Vilchis, M.A. (2010). Panorama de los ataques en los modelos 3D. **Telematique,** 51-63.

MORRIS M., Mano (1994). **Computer system architecture.** Prentice Hall.

MULLEN, Tony (2009). Mastering Blender. SYBEX Serious Skills.

PARDINES, L.I. (2005). **Técnicas paralelas aplicadas a optimización no lineal en sistemas de memoria distribuida.** Universidad Santiago de Compostela.

RANKIN, Kyle (2008). Knoppix Hacks Tips and Tool for Using the Linux Live CD to Hack. O'Reilly.

SÁNCHEZ, J. (2004). Análisis, diseño e implementación de una solución económica y eficiente para visualización a alta resolución de imágenes científicas de simulaciones climáticas para su uso en la ESPOL (Tesis de Ingeniería, Facultad de Ingeniería Eléctrica y Computación, Escuela Superior Politécnica del Litoral).

SCHROEDER, M.D. (1993). A state-of-the-art distributed system: computing with BOB. In **Distributed Systems** (2nd Ed.), S. Mullender, Ed. Acm Press Frontier Series. ACM Press/Addison-Wesley Publishing Co., New York, 1-16.