Trazado de Rayos en Redes de Estaciones de Trabajo*

F. José Alfaro, Pascual González, José L. Sánchez

Universidad de Castilla-La Mancha
Escuela Politécnica Superior
Campus Universitario, 02071 Albacete, España
{falfaro,pgonzalez,jsanchez}@info-ab.uclm.es

Resumen

La gran relevancia que están adquiriendo las redes de estaciones de trabajo como plataformas de ejecución paralela, debido a su buena relación coste/prestaciones, está motivando numerosos trabajos sobre el comportamiento de diferentes algoritmos paralelos en este tipo de arquitecturas.

En este trabajo presentamos un análisis de las nuevas situaciones que se pueden presentar al intentar extrapolar algunas de las ideas utilizadas en otro tipo de máquinas paralelas, en nuestro caso con un esquema de memoria compartida, a este entorno. Para ello se ha implementado una versión paralela del algoritmo de trazado de rayos, utilizando MPI, que se apoya en la idea de subdivisión del espacio de la imagen. Concretamente, se ha estudiado el comportamiento de este algoritmo en sistemas total y parcialmente dedicados al procesado de dicho algoritmo, y considerando o no homogeneidad en las características físicas de los elementos de procesamiento.

1 Trazado de rayos: alternativas de aceleración

El algoritmo de trazado de rayos es utilizado frecuentemente dentro de la visualización de escenas tridimensionales, obteniendo imágenes de gran calidad (foto-realistas). Este algoritmo considera que la pantalla del ordenador está localizada entre un observador imaginario y la escena, y dicho observador lanza una serie de rayos luminosos hacia la escena. La versión inicial de este algoritmo [1] pretende analizar, para cada uno de dichos rayos, su intersección con todos los objetos de la escena. Si un rayo choca con un objeto, entonces el algoritmo propone generar otra serie de rayos que permitirán calcular los efectos de sombreado, reflexión y refracción. Como puede verse, esta versión inicial del algoritmo implica un alto coste computacional y requiere, por tanto, de nuevas propuestas que obtengan una aceleración del mismo.

Actualmente existen multitud de algoritmos de aceleración que pretenden reducir el coste computacional del algoritmo inicial aplicando ciertas propiedades de las escenas o de las imágenes generadas. Estas propiedades pueden expresarse mediante el concepto básico de "coherencia". Podemos encontrar cuatro tipos de coherencia [2, 3]: coherencia de objetos

*Este trabajo está soportado en parte por la CICYT, dentro del proyecto TIC97-0897-C04-02

[4, 5, 6, 7, 8], coherencia de rayos [9, 2, 8], coherencia de imagen [10] y coherencia de fotograma [11, 12].

Pero si realmente se desea reducir el tiempo necesario para generar una imagen utilizando el trazado de rayos, junto a las alternativas anteriormente propuestas, es necesario tener en mente las posibilidades de paralelismo que este algoritmo nos ofrece. El trazado de rayos es intrínsecamente paralelo ya que cada rayo puede ser evaluado independientemente del resto.

Al igual que sucede con otros algoritmos de generación de imágenes [13], en la paralelización del trazado de rayos existen dos grandes alternativas: subdivisión del espacio de la imagen y subdivisión del espacio de los objetos. La primera de ellas, distribuye los pixel de la pantalla a analizar entre los diferentes procesadores y la segunda realiza una distribución de los objetos de la escena entre dichos procesadores. Esta última propuesta es principalmente utilizada en la paralelización del trazado de rayos en arquitecturas MIMD con memoria distribuida.

En el primer esquema, subdivisión del espacio de la imagen, la totalidad de los pixel asociados al dominio de la imagen se descompone en subdominios. Tras esto, cada uno de ellos es asignado y calculado independientemente por un procesador diferente. Sin embargo, para poder procesar de modo independiente cada parte de la imagen, cada procesador debe tener acceso a toda la información referente a la totalidad de los objetos asociados a la escena a representar. El modo más simple para facilitar el acceso a la totalidad de la escena consiste en duplicar completamente la estructura de objetos asociada a la escena en la memoria local de cada procesador [14]. En cualquier caso, en la actualidad existen otras alternativas que utilizan computadores paralelos de memoria virtual compartida [15]. Estas nuevas propuestas garantizan la accesibilidad a la totalidad de la información a la vez que simplifican el proceso de acceso por parte de todos los procesadores a la totalidad de la estructura de datos asociada a la escena.

En la segunda alternativa, subdivisión del espacio de los objetos, el espacio de los objetos es subdividido y almacenado en la memoria local de cada procesador. Este proceso de subdivisión reduce la cantidad de memoria necesaria para procesar una escena ya que ésta está físicamente almacenada en la memoria local de los diferentes procesadores. Sin embargo, complica el diseño de los algoritmos paralelos a la vez que aumenta de modo considerable las necesidades de comunicación entre los diferentes procesos ya que no son capaces de calcular completamente el color de algunos de los pixel que le han sido asignados. Dentro de los diferentes algoritmos que realizan este tipo de distribución podemos encontrar dos grandes alternativas de tratar el proceso de comunicación entre procesos: basado en el movimiento de objetos o basado en el movimiento de rayos. La primera de ellas realiza un envío de la parte de la escena necesaria para seguir procesando los rayos, al procesador que los solicita. Su funcionamiento se apoya en la idea de la gestión de memoria cache. En cambio la segunda simplemente realiza un envío del rayo individual al procesador donde se encuentra la parte de la escena que atraviesa el rayo dado.

Dentro de la extensa bibliografía existente que aborda la paralelización del trazado de rayos podemos encontrar bastantes propuestas en arquitecturas de tipo MIMD que utilizan la distribución del espacio de los objetos [16, 3, 17]. En todo caso, como apuntan Badouel, Bouatouch y Priol [15], esta alternativa tiene importantes inconvenientes.

En primer lugar esta aproximación, basada en paso de mensajes, es más difícil de diseñar y su eficiencia decrece conforme se incrementa el número de procesadores que intervienen

en la paralelización. En segundo lugar, varias regiones pueden compartir el mismo objeto implicando, por tanto, una cierta replicación de la información y una repetición de los cálculos de intersección con dichos objetos cuando un rayo se mueve desde una parte de la escena a otra que comparte el mismo objeto. Finalmente, el balanceo de la carga es más difícil y costoso ya que los criterios de descomposición de la escena no son en ningún caso obvios si lo que se pretende es que dicha subdivisión consiga un equilibrado balanceo de la carga asignada a cada procesador.

Por último, el gran auge que tienen las estaciones de trabajo (cada vez más potentes) y las redes de interconexión (cada vez más rápidas) están haciendo que en la actualidad estén surgiendo nuevas propuestas basadas en estas nuevas alternativas arquitectónicas [18, 19, 20]. Dichas propuestas intentan trasladar las diferentes ideas aportadas anteriormente dentro del campo de los algoritmos paralelos a este renaciente ámbito de actuación. En esta línea de trabajo nuestro artículo quiere mostrar una implementación del trazado de rayos en este tipo de arquitecturas. Con ello pretendemos reflejar algunos de los problemas que pueden aparecer en este tipo de sistemas, problemas que no se presentan generalmente en otros entornos de trabajo.

Por ello, en las siguientes secciones describiremos el algoritmo utilizado y presentaremos algunos resultados que permitan observar ciertos problemas asociados con estos entornos, muchos de ellos derivados de las posibilidades de utilización de máquinas heterogéneas y que simultáneamente están cargadas por otros procesos.

2 Algoritmo Paralelo

Nuestra propuesta de paralelización está basada en la utilización de la subdivisión del espacio de la imagen, por tanto consideramos que todos los procesadores tienen accesible la información de la totalidad de la escena. En este caso cada procesador es capaz de trazar de modo independiente los rayos que se le asignen. La elección de esta aproximación está basada en la eficiencia de esta alternativa con respecto a la subdivisión del espacio de los objetos [15], la cual implica una gran cantidad de comunicación entre los diferentes procesadores y, por tanto, hace que el tiempo debido a dicha comunicación sea tan elevado que en algunos casos el algoritmo paralelo pueda comportarse incluso peor que la versión secuencial.

Para implementar la versión paralela nos hemos apoyado en la librería desarrollada por Wilt [21] (Object Oriented Ray Tracing - OORT). Esta librería implementa en C++ todas las clases necesarias para realizar la generación de la imagen asociada a una escena cualquiera. Los objetos de la misma pueden definirse utilizando tanto representaciones de tipo CSG (Constructive Solid Geometry) como B-rep (Boundary Representation). Aunque en este último caso se ha mejorado la representación de los objetos al añadir un nuevo tipo de primitiva que permite realizar un suavizado de los polígonos que determinan el contorno de los objetos.

Por otra parte, la librería que hemos usado utiliza dos técnicas de aceleración. Por un lado, apoyándose en la idea de la coherencia de objetos, utiliza una estructura de descomposición de la escena guiada por los objetos en ella contenidos. En este caso crea una jerarquía de volúmenes envolventes que permiten guiar en cierta medida el trazado de los diferentes rayos a través de la escena. La creación de la estructura se basa en los trabajos de Goldsmith y

Salmon [5] y como elemento envolvente utiliza cajas alineadas con los ejes. Por otra parte el recorrido de la estructura se optimiza utilizando el algoritmo propuesto por Kay y Kajiya [4]. El otro tipo de aceleración se apoya en la coherencia de rayos y está asociada al tratamiento de los rayos de sombra ("cache shadow"). De este modo, pretende aprovechar la información recogida por un rayo de sombra para acelerar el trazado del siguiente. Para ello, cada luz almacena la información del objeto con el que chocó el último rayo de sombra que se dirigió a dicha luz. Esta última técnica de aceleración ha condicionado en cierta medida el criterio de distribución establecido a la hora de implementar la idea del balanceo entrelazado.

Para la paralelización hemos usado las primitivas de comunicación que nos proporciona el estándar MPI 1.0 en su implementación MPICH 1.0.13 [22], situando un proceso en cada máquina. Hemos usado tanto funciones de comunicación punto a punto entre dos procesos, como funciones de comunicación colectiva entre todos los procesos a la vez.

Nuestro algoritmo paralelo trata de explotar al máximo el potencial de las redes de estaciones de trabajo. Usamos un sistema maestro/esclavo, donde el proceso maestro se encarga de distribuir el trabajo entre los procesos esclavos. Éstos generan la parte de la escena que se les ha asignado y devuelven los resultados obtenidos al proceso maestro. Para hacer la paralelización hemos usado las librerías de MPI [23, 24] con un modelo computacional tipo MIMD.

Dentro de las tareas propias del algoritmo paralelo tal vez la más crítica es el establecimiento, por parte del proceso maestro, de la distribución de la carga entre los diferentes procesadores. Como ya se ha indicado anteriormente, el esquema seleccionado para realizar dicha distribución se basa en la subdivisión del espacio de la imagen. En este caso existen dos grandes alternativas: asignar a cada procesador un bloque de pixel consecutivos o asignar pixel distribuidos uniformemente por toda la pantalla. En este último caso también podemos pensar en hacer una distribución de tal manera que puntos vecinos sean tratados siempre por procesadores distintos, con lo cual la carga asociada a cada proceso será similar, pero ese tipo de distribución hace totalmente ineficiente la optimización de shadow cache propuesta en la librería OORT. Por ello nuestra alternativa de distribución (reparto entrelazado) consiste en repartir líneas horizontales de pixel en vez de pixel independientes.

De este modo, si tenemos F líneas a trazar en la pantalla, y usamos el reparto entrelazado entre P procesos, tendremos que la línea f de la pantalla la va a trazar el proceso $f \ mod \ P$, con lo que le damos a cada proceso líneas de todas las zonas de la pantalla, tanto de las áreas más pobladas de objetos, que serán más complejas de trazar y por lo tanto llevarán más tiempo, como de las más rápidas de trazar. Así, en una escena realista que no estará igualmente poblada, todos los procesos tendrán una carga de procesamiento similar.

Una vez que el proceso maestro ha calculado cuántas líneas le corresponden trazar a cada proceso, les informa a éstos para que preparen el espacio necesario. Para esta comunicación hemos utilizado la función de comunicación colectiva MPI_Scatter que ofrece MPI para distribuir un bloque entre un grupo de procesos, dando a cada uno de ellos un bloque del tamaño que se indique.

Cuando todos los procesos saben las líneas que deben trazar, empiezan a trabajar trazando el intervalo que se les ha asignado. Los procesos esclavos trazan sus líneas de forma simultánea sin interferencia entre ellos, ya que los rayos trazados son independientes y todos los procesos conocen de forma completa la escena. Cuando un proceso termina de tratar su área envía al proceso maestro los resultados que ha obtenido y termina su ejecución.

Cuando el proceso maestro termina de distribuir entre los procesos esclavos los intervalos que deben trazar cada uno, se queda esperando a que vayan terminando los procesos trazadores y le envíen los resultados que han obtenido. Este envío de resultados al proceso maestro se hace con las funciones de MPI para comunicación punto a punto MPI_Send/MPI_Recv. Se usa comunicación punto a punto y no comunicación colectiva porque no todos los procesos van a terminar al tiempo, y es más práctico que el proceso maestro vaya recibiendo los resultados conforme finaliza su trazado cada proceso. El maestro va reuniendo los datos que le envían los otros y poniendo las filas resultado en orden, para generar la imagen final.

3 Análisis de resultados

En este apartado analizamos el comportamiento del algoritmo en dos tipos de sistemas. El primero de ellos es un sistema homogéneo y sin carga, esquema que como veremos se comporta como un sistema masivamente paralelo con memoria compartida típico. Por otra parte, en el segundo apartado se analizará el efecto que tiene sobre los algoritmos anteriores, el hecho de que el sistema esté formado por máquinas de distinto tipo o podamos encontrar algunas de ellas simultáneamente cargadas procesando otros trabajos.

De las escenas utilizadas para comprobar el comportamiento del algoritmo hemos seleccionado una escena simple en la que los objetos están fuertemente concentrados en el centro de la misma. Esta escena nos permitirá ver claramente las ventajas e inconvenientes de las diferentes alternativas analizadas.

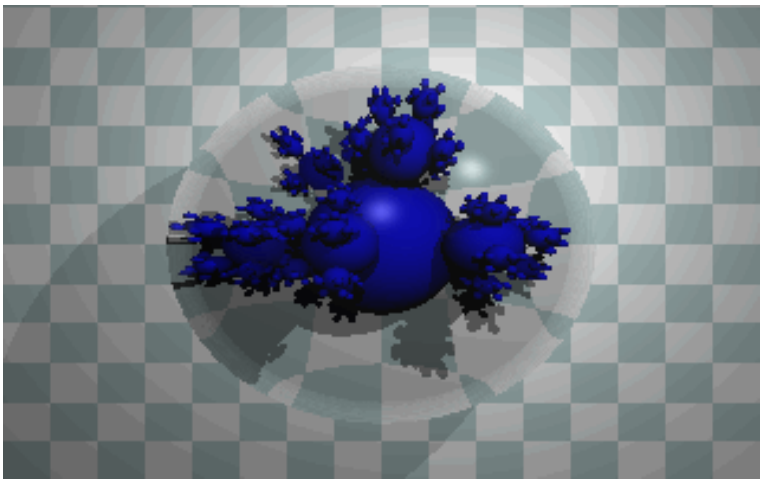

Figura 1: Escena trazada

3.1 Sistemas homogéneos y sin carga

Hemos usado una red de estaciones de trabajo formada por 24 máquinas Pentium a 166 MHz con 16 Mb de memoria RAM, bajo el sistema operativo Linux en su versión Slackware con el núcleo 2.0.0.

En la figura 2 vemos el comportamiento de las dos técnicas típicas de distribución. Este comportamiento es similar al que tendrían en un sistema con memoria compartida. La distribución por bloques se ve afectada en gran medida por la distribución de los objetos en la escena no siendo recomendable en escenas que, como ésta, tengan una distribución no uniforme. En cambio el método entrelazado consigue una distribución ciertamente buena pues cada proceso debe analizar líneas distribuidas uniformemente por toda la pantalla y por tanto la carga que se le asigna es similar.

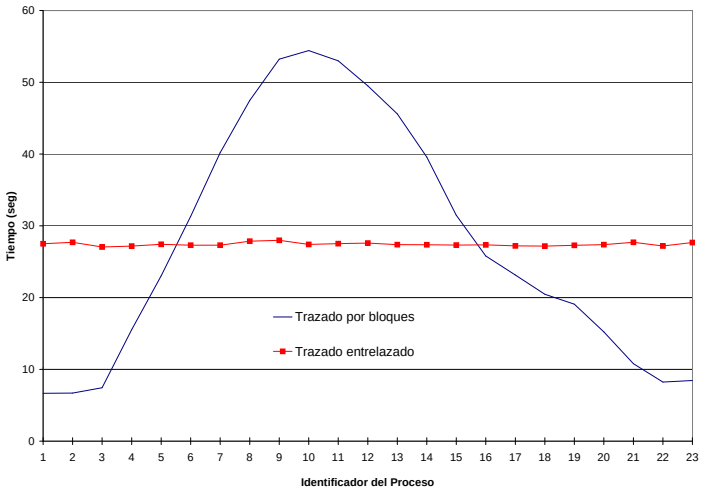

Figura 2: Tiempo de trazado obtenido por cada máquina

Junto a este análisis hemos querido ver el comportamiento del algoritmo conforme aumente el número de procesadores que procesan la escena. En la figura 3 puede verse el resultado obtenido por el algoritmo con el sistema de reparto entrelazado. En este caso, junto al tiempo total de procesado del algoritmo (tiempo total de simulación), hemos separado éste en sus componentes principales para ver de manera más detallada su evolución. Como puede apreciarse en la figura 3 hemos tenido en cuenta dos datos básicos: tiempo para establecer los procesos mediante MPI y tiempo de trazado. El primero de ellos, es el tiempo que las funciones de la librería de MPI tardan en levantar de forma remota los procesos en todas las máquinas utilizadas. El otro tiempo es el debido al proceso de trazado típico. En este caso hemos representado el tiempo del proceso que terminó en último lugar pues es el que condiciona el tiempo real de procesado.

Si analizamos el comportamiento de los diferentes tiempos que aparecen reflejados en la figura 3 podemos observar que la incorporación de procesadores hace disminuir de modo progresivo el tiempo de trazado al tener que analizar cada proceso un menor número de líneas. Esta tendencia es muy acusada hasta 10 procesos, empezando después una caída mucho menor.

Junto a esta tendencia típica del tiempo de trazado con respecto al número de procesos, es interesante resaltar que el otro tiempo, tiempo de establecer los procesos, se incrementa conforme crece el número de máquinas que participan en la distribución. Esta tendencia hace que con 23 procesadores el tiempo de trazado y el tiempo de establecer procesos sea

muy similar. Por ello, vemos que con aproximadamente veinticinco procesadores el tiempo de levantar los procesos puede ser superior a la ganancia obtenida con la disminución del tiempo de trazado lo cual es un serio problema. En este caso parece claro que para conseguir un mayor grado de paralelización sería necesario mejorar las primitivas de comunicación que nos ofrece MPI.

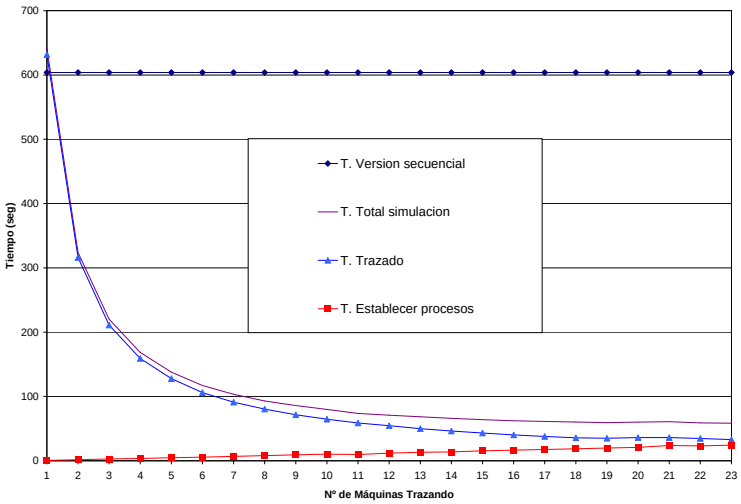

Figura 3: Resultados temporales en función del número de máquinas tranzando

3.2 Sistemas heterogéneos o con carga

Hasta ahora sólo hemos contemplado máquinas de similares características y dedicadas en exclusividad al proceso de trazado, pero esta idea además de cara es poco realista. Lo normal es que en la red de estaciones de trabajo haya distintos tipos de máquinas y otros usuarios ejecutando en ellas diversos tipos de aplicaciones. Esto hará que cada máquina tenga un potencial distinto y soporte una carga adicional que será diferente en cada una de ellas, haciendo que la línea recta de la figura 2 desaparezca siendo ahora como la que muestran las figuras 4 y 5 (reparto equitativo).

Para solucionar este problema debemos medir tanto la carga actual de cada máquina como su potencia relativa respecto a la más rápida [25]. Esta ponderación hará que a cada máquina se le asigne un número diferente de líneas en función de su capacidad disponible.

En este caso surge otro problema potencial, ya que es posible que el reparto de las líneas asignadas a cada proceso sea tal que al proceso que menos líneas se le asignen no tenga dichas líneas distribuidas de manera uniforme y por tanto pueda producirse un nuevo balanceo incorrecto. Este problema puede reducirse si el reparto entrelazado en vez de tener como referencia la totalidad de la pantalla tiene como referencia una ventana de menor tamaño que vamos desplazando a través de la pantalla, de tal forma que las líneas asignadas a cada proceso estén mejor distribuidas dentro de la pantalla.

En las figuras 4 y 5 puede verse el comportamiento de la alternativa que hemos descrito anteriormente (reparto proporcional). Como podemos observar los picos producidos en la

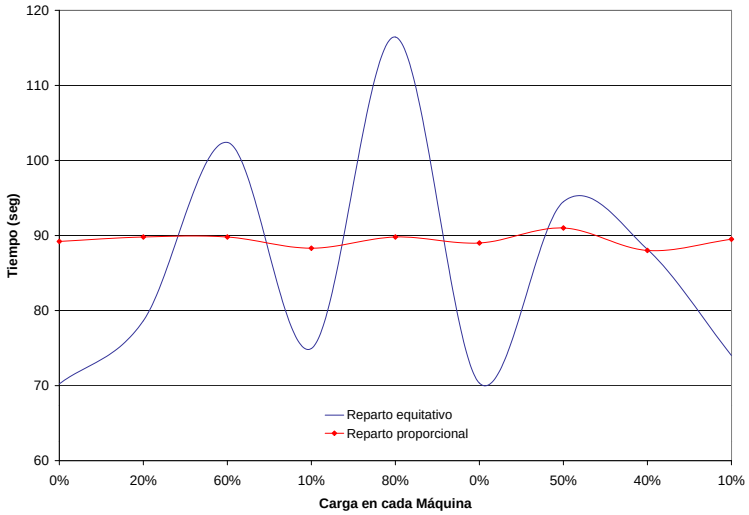

Figura 4: Tiempo de rendering con 10 máquinas con diferente carga

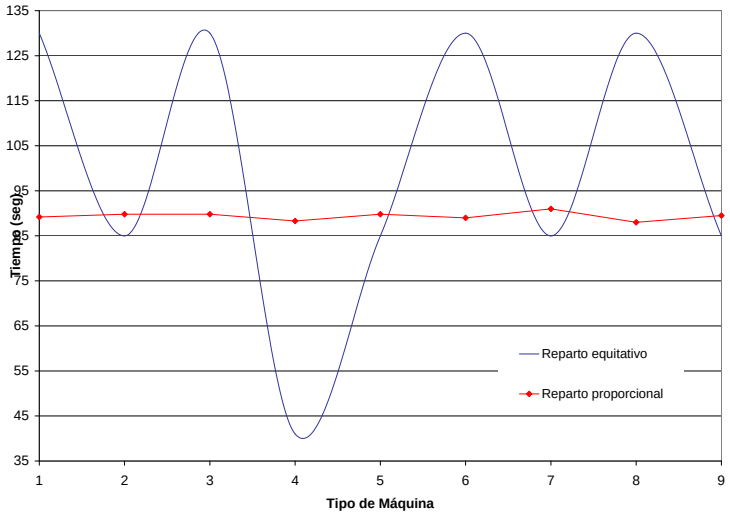

Figura 5: Tiempo de rendering usando máquinas con distintas características

versión utilizada por los sistemas homogéneos y sin carga (reparto equitativo) se reducen bastante obteniendo un mejor balanceo de la carga y un más adecuado aprovechamiento del potencial de las máquinas.

4 Conclusiones

En este trabajo hemos analizado el comportamiento de las versiones paralelas del algoritmo de trazado de rayos que se apoyan en esquemas de subdivisión del espacio de la imagen. Por una parte se han presentado las ideas básicas del algoritmo que ha sido implementado y las características de la librería en la que nos hemos apoyado para su desarrollo.

Por otra parte, se ha analizado el comportamiento del algoritmo en diferentes entornos: sistemas homogéneos y sin carga; sistemas heterogéneos y/o con carga. En este análisis hemos podido ver como aquellas ideas aplicables con éxito en sistemas de memoria compartida se comportan también muy bien en sistemas homogéneos pero sin carga. Junto a esto, hemos podido comprobar que dichas soluciones tienen un peor comportamiento en otro tipo de sistemas en los que las máquinas tengan una potencia distinta o su capacidad disponible sea diferente. En este último caso se han apuntado algunas ideas que permiten adaptar los algoritmos de distribución de la carga iniciales a este tipo de entornos heterogéneos o cargados, sistemas muy comunes en las redes de estaciones de trabajo.

Referencias

[1] Whitted, T.: An Improved Illumination Model for Shaded Display. Communications of the ACM, **23**, no. 6, (1980), 343–349

[2] Ohta, M., Maekawa, M.: Ray Coherence Theorem and Constant Time Ray Tracing Algorithm. Computer Graphics, 1987, pp.303-314.

[3] Green, S.A., Paddon, D.J.: Exploiting Coherence for Multiprocessor Ray Tracing. IEEE Computer Graphics & Applications, **4**, No. 10, (1989), 12–26

[4] Kay, T., Kajiya, J.: Raytracing complex scenes. Computer Graphics, **20**, No. 4, (1986) 269–278

[5] Goldsmith, J., Salmon, J.: Automatic creation of object hierarchies for ray tracing. IEEE Computer Graphics & Applications, **7**, No. 5, (1987), 14–20

[6] Amantides, J., Woo, A.: A fast voxel traversal algorithm for ray tracing. In EUROGRAPHICS '87, MARECHAL, G. (ed.), (1987), 3–10

[7] Samet, H.: Implementing Ray Tracing with Octrees and Neighbor Finding. Computer & Graphics, **13**, No. 4, (1989), 445–460

[8] González, P., Gisbert, F.: Object and Ray Coherence in tha Optimization of the Ray tracing Algorithm. Proc. Computer Graphics International, CGI'98. IEEE Computer Society Press, June 1998, pp. 264-267.

[9] Heckbert, P., Hanrahan, P.: Beam Tracing Polygonal Objects. Computer Graphics, Vol. 18, No 3, 1984, pp. 119-127.

[10] Weghorst, H., Hooper, G., Greenberg, D.: Improved Computational Methods for Ray Tracing. ACM Trans. on Graphics, **3**, No. 1, January (1984), 52–69

[11] Glassner, A.S.: Spacetime ray tracing for animation. IEEE Computer Graphics & Applications, **8** ,No. 3, March (1988), 60–70

[12] Gröller, E., Purgathofer, W.: Using temporal and spatial coherence for accelerating the calculation of animation sequences EUROGRAPHICS'91, Post, F.H., (1991), 103–113

[13] Whitman, S.: Multiprocessor Methods for Computer Graphics Rendering. Jones and Bartlett Publishers, (1992)

[14] Naruse et al.: Sihgt: A Dedicated Computer Graphics Machine. Computer Graphics Forum, **6**, No. 4, December (1987), 327–334

[15] Badouel, D., Bouatouch, K., Priol, T.: Distributing Data and Control for Ray Tracing in Parallel. IEEE Computer Graphics and Applications, (1994), 69–77

[16] Kobayashi, et. al.: Load Balancing Parallel Ray-Tracing System Based on Constant Subdivision. The Visual Computer, **4**, (1988), 197–209

[17] Priol, T., Bouatouch, K.: Static Load Balancing for a Parallel Ray Tracing on a MIMD Hypercube. The Visual Computer. **4**, No. 12, March (1989), 109–119

[18] Reinhard, E., Chalmers, A.: Message Handling in Parallel Radiance. Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proc 4th European PVM/MPI Users Group Meeting, Bubak, M., Dongarra, J., Wasniewski, J. ed. Lecture Notes in Computer Science. Cracow, Poland, Nov. 1997, pp.486-493.

[19] Davis, T.A., Davis, E.W.: Rendering Computer Animations on Network of Workstations. In Proc of the IPPS/SPDP, IEEE Computer Society Press, April 1998, pp. 726-730.

[20] Haimes, R., Jordan, K.E.: Using PVM and MPI for Co-processed Distributed and Parallel Scientific Visualization. Parallel Distributed Processing ROLIM, J. ed. Lectures Notes in Computer Science, Springer 1998, pp. 1098-1105.

[21] Wilt, N.: Object-Oriented Ray Tracing in C++. John Wiley & Sons, Inc. 1994

[22] W. Gropp and E. Lusk.: User's Guide for mpich, a Portable Implementation of MPI. Available by ftp from info.mcs.anl.gov in /pub/mpi/guide.ps.Z, November 1994.

[23] MPI Forum: "MPI: a message interface standard". International Journal of Supercomputer Applications, 8(3/4), 1994. Special issue on MPI.

[24] J. Dongarra, M. Snir, S. Otto, S. Huss-Lederman, D. Walker.: MPI: The Complete Reference. The MIT Press. (1996).

[25] F. Alfaro, J.A. Gallud, J.L. Sanchez.: A Function to Dynamic Workload Allocation in Distributed Applications. Fourth EuroPVM/MPI.Cracow (Poland) (1997).