ZeBrA-Core: Una Herramienta para la Generación de Imágenes Fotorrealistas Usando POV-Ray en Ambientes de Computación Distribuida

ZeBrA-Core: a Computer Tool to Render Photorrealistic Images Using POV-Ray in Distributed Computing Environments

Diego A. Bedoya Salazar
Departamento de Sistemas, Facultad de Ingeniería

Carolina Escobar Cadavid
Departamento de Sistemas, Facultad de Ingeniería

Jorge I. Zuluaga Callejas
Grupo de Física y Astrofisica Computacional, Instituto de Física.

Universidad de Antioquia (Colombia)
{alejobedoya, caesca@gmail.com}, {jzuluaga@udea.edu.co}

Resumen

Presentamos una herramienta que implementa la partición y distribución de imágenes fotorrealistas generadas mediante técnicas de ray-tracing con POV-Ray sobre plataformas de computación distribuida. La síntesis de la imagen se divide en partes iguales y se encarga la tarea de generación de cada porción a instancias de POV-Ray, que se ejecutan concurrentemente en recursos típicos de un ambiente de computación distribuida. Las subimágenes generadas se transfieren a un recurso central y se fusionan, usando un módulo de ImageMagick para formar la imagen completa. La herramienta desarrollada automatiza el proceso de división, aglomeración, asignación, generación y fusión final de la imagen; y tiene el potencial para funcionar en diversos ambientes de computación distribuida. Los resultados de su aplicación en un cluster demuestran su capacidad para reducir los tiempos de generación de imágenes de muy alta resolución, con una eficiencia muy alta y un costo muy bajo de uso de red.

Palabras clave: *Generación de imágenes: técnicas de distribución y paralelismo, POV-Ray, computación distribuida, granjas de generación de imágenes.*

Abstract

Introducing a tool which implements the partition and distribution of photorealistic images generated using ray-tracing techniques by POV-Ray on distributed computing platforms. Rendering of the image is divided in equally spaced parts and then submits them to POV-Ray instances, running concurrently on a set of distributed resources. Individually images so generated are then joined using an ImageMagick module to obtain the complete image, in a central resource. This computer tool automates all the processes involved: partitioning, agglomeration, mapping and construction, and has the potential to work in different distributed environments. The results of the application of this tool in a cluster proves its capability to highly reduce the rendering times for very high resolution images with high efficiency and at very low cost in terms of network occupation.

Keywords: *Rendering: distribution techniques, parallel processing. POV-Ray, distributed computing, render farms.*

1. Introducción

La generación de imágenes fotorrealistas con características 3D (en adelante *rendering 3D*), es hoy un problema muy común en sectores como el diseño gráfico, el diseño industrial, la publicidad y el cine. Basta reconocer como un ejemplo inmediato la popularidad que en años recientes han tenido películas animadas con simulaciones ultra realistas de personajes y escenas complejas [2], o el uso de la síntesis de imágenes de alta calidad y resolución para representar de forma realista sitios o eventos no existentes, difíciles de acceder, en negocios como diseño industrial, propiedad raíz, publicidad o cultura.

En general el rendering asistido por computador de escenas tridimensionales comienza con la descripción matemática de la escena y las propiedades detalladas de los objetos y los medios, sobre y a través de los cuales se propaga la luz. Muy diversas técnicas han sido desarrolladas e implementadas para construir a partir de esa descripción formal, la imagen final de una

escena. Entre las más conocidas se encuentran las técnicas de *ray-casting* [4], *rasterización* (p.e. *scanline rasterization*) [12], radiosidad [9] y *ray-tracing* [9][1].

Entre las técnicas mencionadas, que son usadas con distintos propósitos de acuerdo a las herramientas disponibles o a los requerimientos de tiempo implicados, el *ray-tracing* es quizás aquella con la mayor capacidad para generar representaciones con el más alto realismo fotográfico. La técnica utiliza métodos de Monte Carlo para generar las propiedades de cada píxel en la imagen calculando la trayectoria e interacciones ópticas detalladas de rayos de luz que se propagan desde la cámara hacia las fuentes de luz presentes en la escena [7]. Usando una detallada descripción de las leyes de la óptica que se aplican para determinar las propiedades de los rayos, además de una aproximación más realista al proceso en el que se forma la imagen, el *ray-tracing* logra simular efectos que otras técnicas no logran consiguiendo un realismo fotográfico impactante (ver figura 1).

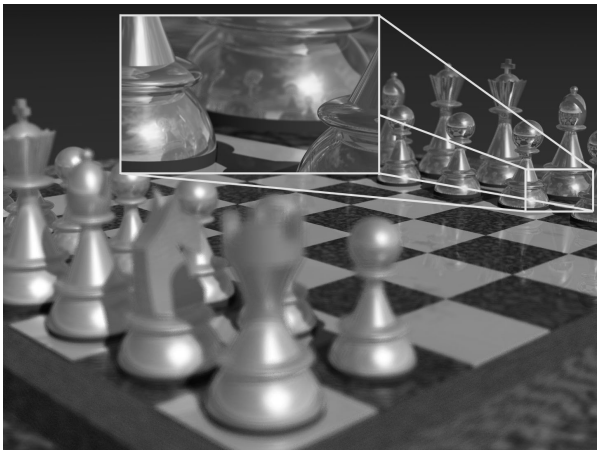


Figura 1. Chess2. Escena de un tablero de ajedrez generada usando ray-tracing con POV-Ray teniendo en cuenta efectos de enfoque. Se muestra el detalle de una parte de la escena (calculada independientemente con los métodos descritos en este trabajo) que muestra el nivel de realismo que consigue el software al tener en cuenta complejos fenómenos ópticos.

El ray-tracing es sin embargo un proceso relativamente lento. La duración de las tareas de rendering con esta técnica depende del nivel de realismo que se desea obtener, y esta determinado por distintos factores. Para aplicaciones que requieran un número muy grande de imágenes como la construcción de animaciones en tres dimensiones, el ray-tracing puede ser menos práctico que otras técnicas de rendering. Sin embargo en otras áreas donde pocas imágenes son requeridas pero un alto nivel de realismo es deseado, el precio más alto se paga cuando se desean obtener imágenes de muy alta resolución. Los tiempos de rendering en estos casos pueden oscilar entre algunas horas y varios días [8].

La popularización de plataformas de computación distribuida (máquinas multiprocesador, clusters y grids de cómputo) en la última década y media, ha permitido aliviar el problema del rendering fotorrealista con técnicas densas como las que usa el *ray-tracing*. Es cada vez menos extraño encontrar soluciones de software y de hardware que permiten la utilización de muchos recursos de cómputo que realizan tareas de rendering complejas de forma cooperativa [6]. Plataformas especializadas son utilizadas por ejemplo en la industria del cine y el entretenimiento pero son menos comunes en el sector del diseño gráfico, el diseño industrial y la publicidad. Los tiempos de generación en estos casos pueden oscilar entre algunas horas y semanas [8].

Pero el desarrollo específico de las que son ya herramientas de software relativamente complejas (motores de rendering) para que puedan correr sobre plataformas distribuidas es un problema todavía muy complejo. Solo herramientas muy sofisticadas y costosas tienen esta capacidad y dependen además de la existencia de plataformas de cómputo igualmente costosas. La propuesta central de este trabajo es precisamente adaptar una muy bien conocida y robusta herramienta de código abierto, POV-Ray [9], para generar imágenes de muy alta resolución sobre plataformas de computación distribuida, con un mínimo esfuerzo de desarrollo. Existen versiones estables de POV-Ray para usar en máquinas paralelas cuyo desempeño ha sido probado y demostrado en múltiples aplicaciones [3]. Sin embargo el desarrollo de estas versiones ha exigido un esfuerzo importante de reescritura o adaptación de parte del código del programa. Esta condición causa que las versiones paralelas se retrasen normalmente respecto a las últimas versiones del programa que incluyen características y efectos cada vez más novedosos. Adicionalmente la ejecución de POV-Ray en paralelo (al menos en las versiones existentes) tiene unas exigencias de software y de hardware específicas y una tolerancia relativamente baja a las fallas en tiempo de ejecución.

El método y la herramienta desarrollada en el marco de este trabajo intentan resolver algunas de estas limitaciones. El desarrollo de la herramienta no requiere la modificación del código fuente de POV-Ray y trabaja en una capa superior a la del programa en el sistema. Se puede, por esa misma razón, usar las nuevas versiones del programa sin esperar al desarrollo de versiones paralelas. La más importante característica es que puede usarse sobre cualquier plataforma de cómputo distribuido, e inclusive en esquemas relativamente simples de interconexión de

1. Es común, por lo novedoso de esta área, que muchos vocablos no tengan todavía su contraparte en castellano. El uso de los términos *render*, *ray-tracing*, *ray-casting*, *rasterización* en el texto obedece a motivaciones enteramente prácticas.

recursos.

El presente artículo se estructura de la siguiente manera: en la sección 2 presentamos una descripción general de los problema y retos que implica el rendering de imágenes de muy alta resolución. En la sección 3 se describen el método de particionado y la estrategia de distribución utilizado en este trabajo para realizar esta tarea de forma distribuida. La sección 4 presenta algunos detalles del diseño de la herramienta de software desarrollada para realizar las pruebas básicas del método. En la sección 5 se presentan los resultados de las pruebas ejecutadas sobre un Cluster Beowulf y su comparación con pruebas similares realizadas con herramientas paralelas. Finalmente en la sección 6 se presenta una discusión general sobre los resultados, las conclusiones y perspectivas del trabajo y se realiza una reflexión sobre las aplicaciones prácticas que podría tener la herramienta.

2. Los problemas

El problema de la generación de imágenes en plataformas distribuidas ha sido estudiado exhaustivamente en la literatura ([6], [10], [3], [11] y referencias contenidas en ellos). Muy diversos esquemas de particionado, estrategias de distribución y de balanceo de carga han sido diseñadas y aplicadas en diversas condiciones para el desarrollo, principalmente, de versiones paralelas de reconocidos programas de rendering.

En general puede decirse que el rendering distribuido enfrenta dos problemas básicos: el particionado y el balanceo de carga [3]. El primero de ellos se relaciona con la manera como la tarea de rendering es dividida en subtareas que pueden ejecutarse concurrentemente. El balanceo de carga, en este contexto, se refiere a la compleja tarea de definir los parámetros de particionado que producen subtareas de tamaño diverso que pueden ejecutarse en tiempos comparables en los recursos de cómputo disponibles [10], [5].

Cuando se consideran estrategias y mecanismos de rendering distribuidos como los considerados en este trabajo, otros tres problemas deben ser abordados: la aglomeración, la asignación y la fusión final de porciones de imagen generadas. La aglomeración es el proceso de reunir en un número menor de trabajos de cómputo las subtareas más pequeñas definidas en el proceso de particionado. La asignación implica definir los mecanismos específicos para entregar las tareas de rendering a los recursos en la plataforma, monitorear su ejecución y recoger los resultados de la misma. La fusión, un problema exclusivo de los métodos que recurren a la generación independiente de porciones de imagen, implica la construcción de una imagen individual a partir de la reunión de las porciones generadas por cada tarea de rendering.

En la siguiente sección se describen los métodos y estrategias específicas que fueron utilizados para resolver los problemas enumerados en el contexto de este trabajo.

3. La solución

3.1. Particionado

El método de rendering utilizado por POV-Ray se presta idealmente para el diseño de muy diversos esquemas de particionado. Como se explico en la sección 1, POV-Ray calcula independientemente las propiedades de cada píxel de la imagen haciendo propagar la luz desde el plano sobre el que ella se forma hacia los objetos y fuentes de luz que se encuentran en la escena. La información obtenida de cada píxel es en general independiente de la obtenida para los demás (exceptuando quizás algunos píxeles vecinos en los casos de efectos y técnicas como el antialiasing [3]). Esto permite que porciones enteras de imagen puedan generarse en instancias independientes de ejecución. El primer y más natural esquema de particionado que puede considerarse es el de dividir la imagen en porciones de geometría simple (rectángulos) que no se superponen.

Dos esquemas globales de división pueden considerarse: una división total de la imagen (a lo ancho y a lo alto) en nxm porciones (n divisiones verticales y m horizontales) o una división por franjas (strips) en la que la imagen se fracciona solo en una dirección (n franjas horizontales o m franjas verticales). Esquemas más generales de división pueden considerarse combinando estos dos esquemas básicos.

Suponiendo que tenemos una imagen que representa una escena más o menos homogénea el tiempo de cómputo total escala como el número de píxeles en la escena R=HxW, siendo H la “altura” de la imagen en pixels y W su “ancho”. Si el tamaño a lo ancho y a lo alto se modifica en un mismo factor f ,(H' = fH, W' = fW) el tiempo de procesamiento crecerá como la resolución R' = f2 HxW, es decir en un factor f2. Este crecimiento es responsable del incremento considerable del tiempo de rendering de imágenes de muy alta resolución usando esta técnica. Para el caso por ejemplo de la escena de la figura 1, la imagen de resolución menor que muestra todo el tablero (1440x1080 px) requirió un tiempo de procesamiento aproximado de 17 minutos. El detalle se obtuvo de una imagen 6 veces mayor (8640x6480 px) para la que se invirtió un tiempo de más de 10 horas!

Si se hace una división de la imagen en N porciones la resolución y el tiempo de procesamiento de cada una de ellas se reducirá en un factor 1/N. El tiempo de procesamiento distribuido dependerá del tiempo de procesamiento de cada porción y del número de

procesadores individuales disponibles para realizar la tarea, Np. Si Np < N, las porciones deberán aglomerarse en grupos de N/Np tareas y el tiempo de procesamiento distribuido de la imagen escalará solo como 1/Np. Si N>Np el tiempo escalará con el inverso del número de porciones. Este último comportamiento es siempre el deseado.

Si imponemos una restricción en el número de procesadores disponibles y calculamos el factor de división en alto y/o en ancho (n y/o m respectivamente) para los dos esquemas de división generales, notamos que la subdivisión en franjas es más apropiada para obtener con el mismo número de procesadores una reducción igual del tiempo de procesamiento con un tamaño menor de cada franja al menos en una dirección. Por ejemplo si se cuenta con tan solo Np = 25 procesadores, para asegurar que N < Np en el esquema de división total solo se podrán hacer divisiones de hasta 5x5 porciones. En el esquema de división por franjas se podrá dividir la imagen en hasta 25 franjas horizontales o verticales. La importancia de obtener porciones de imagen más pequeñas en una u otra dirección en este trabajo estriba en el hecho de que la mayoría de las imágenes son no homogéneas (distintas porciones requieren distintos tiempos de procesamiento). Al "atomizar" más la imagen (con un mismo número de procesadores) las regiones que requieren más procesamiento podrán ser identificadas y eventualmente esta información puede ser utilizada para diseñar esquemas de balanceo de carga.

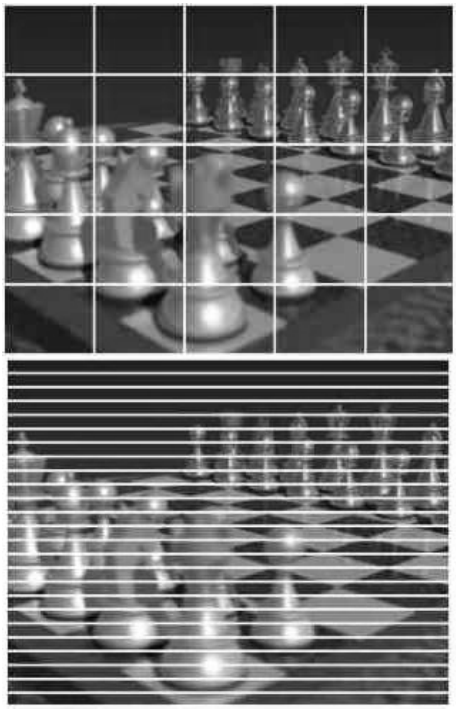


Figura 2. Comparación del nivel de "resolución" del esquema de división total (arriba) y división por franjas para un mismo número de procesadores disponibles.

En este trabajo se ha utilizado un esquema de particionado por franjas con el objeto de garantizar las condiciones especiales mencionadas anteriormente.

3.2. Aglomeración

Existen 2 condiciones en las que la aglomeración se constituye en un problema clave para el rendering distribuido. La primera, muy común, se presenta cuando el número de procesadores disponibles es menor que el número de porciones en las que se divide originalmente la imagen. En este caso la aglomeración garantiza que el rendering pueda siempre realizarse en condiciones variables de disponibilidad de recursos. La segunda condición se presenta cuando los recursos a los que se asignan las tareas tienen capacidad de ejecutar concurrentemente procesos (múltiples procesadores, procesadores con tecnologías como Hyperthreading). En este caso la aglomeración y la ejecución simultanea de tareas que pertenecen a un mismo grupo permitirán un uso más eficiente de esos mismos recursos.

3.3. Asignación

La asignación de tareas en una plataforma de cómputo distribuido a los recursos disponibles en la misma ha sido también un problema ampliamente estudiado en la literatura. Una multitud de herramientas de muy diverso tipo y para diferentes clases de entornos han sido desarrolladas y utilizadas en Clusters y Grids en los últimos 20 años. La estrategia de asignación utilizada en el marco de este trabajo es precisamente aquella de entregar las tareas que resultan de los procesos de particionado y aglomeración a una herramienta independiente.

Esta estrategia reduce el costo de desarrollo de la herramienta de rendering distribuido y garantiza además dos condiciones importantes: primero que los trabajos se ejecutarán obedeciendo los condicionantes de un sistema de colas en lugar de simplemente crear instancias de ejecución sin criterios de disponibilidad de recursos (como sucede con muchos herramientas de ejecución paralela). En segundo lugar el monitoreo y control de los procesos puede realizarse más eficientemente utilizando las herramientas con las que para tal fin cuenta el sistema de asignación de trabajos. Otra de las ventajas de utilizar una herramienta externa para la asignación de trabajos es que prácticamente todas las plataformas de cómputo distribuido cuentan con una herramienta de este tipo; lo difícil es que cuenten con la configuración y mecanismos requeridos para la ejecución de procesos en paralelo.

Finalmente la asignación de trabajos de rendering usando herramientas existentes no pone especiales restricciones sobre la versión del motor de rendering utilizado. Basta que el software este instalado y

propiamente configurado en cada uno de los recursos sobre los que se ejecutarán las tareas individuales para que el proceso completo pueda llevarse a cabo. Esto permite que incluso las últimas versiones del software puedan utilizarse. No es ese el caso de versiones paralelas que deben desarrollarse desde las fuentes del software y que se actualizan a un ritmo menor que el software secuencial.

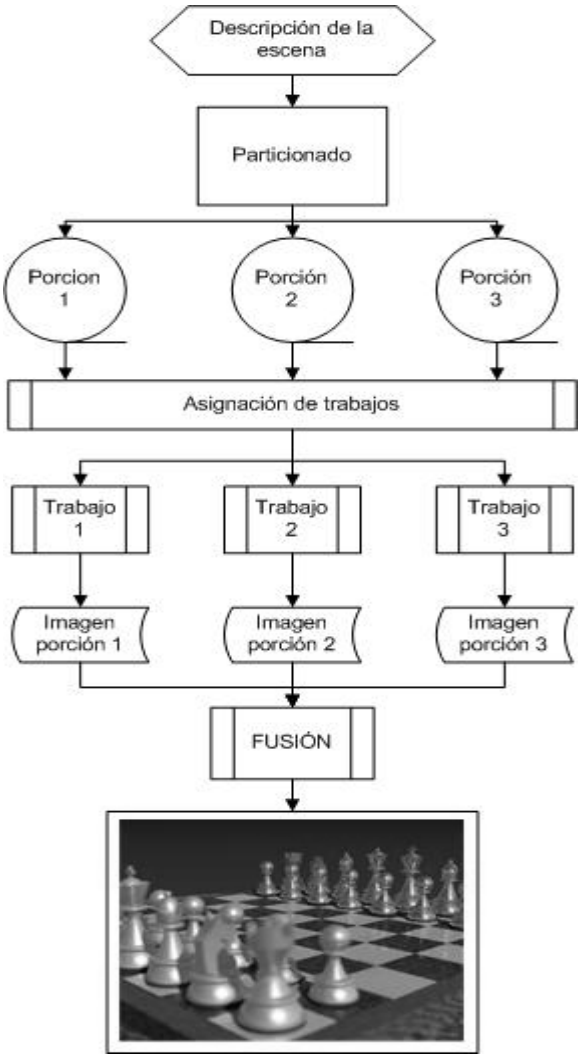


Figura 3. Esquema general del proceso de generación de una imagen en una plataforma distribuida como es concebido en este trabajo.

3.4. Fusión

Como se menciono antes la fusión es la tarea de reunir en una sola imagen las porciones individuales generadas independientemente. El proceso de fusión se da en 2 pasos: adquisición y lectura de las imágenes de las porciones individuales y construcción de la imagen final.

Para la adquisición y lectura de las imágenes de las porciones individuales, el primer condicionante es que los archivos respectivos estén presentes en un sistema de archivos que pueda accederse para operaciones de lectura. De no ser este el caso las imágenes deben transferirse primero a un sistema de archivos local en la máquina donde se realiza la fusión. La construcción de la imagen final se realiza una vez leída la totalidad de las porciones e identificadas las posiciones en las que se ubica cada una en la imagen final.

Para realizar ambas tareas diversas herramientas informáticas pueden utilizarse. Librerías como libpng o ImageMagick pueden ser soluciones apropiadas cuando si se quiere construir una herramienta que funcione automáticamente con una mínima intervención del usuario. En la figura 3 se ilustra gráficamente como cada una de las estrategias y métodos descritos en las subsecciones precedentes se integran para dar producir a partir de un problema de rendering inicial la imagen final de una escena específica.

4. Implementación y herramienta

Para poner a prueba el esquema descrito en la sección anterior se diseño y construyo una herramienta de generación de imágenes usando como motor de rendering POV-Ray. La herramienta que bautizamos **ZeBrA-Core** por el método de partición en franjas y por constituirse en el "corazón" de una versión más elaborada y en desarrollo que contará con una interface gráfica y acceso a través de la Web [1], esta escrita enteramente en perl y cuenta con una interface de línea de comandos y fue desarrollada para el sistema operativo GNU/Linux. La primera versión de **ZeBrA-Core** esta constituida por 5 módulos (zebra-core-strips, zebra-core-procs, zebra-core-launch, zebra-core-stat y zebra-core-build), una biblioteca de rutinas específicas (zebra-core-routines) y un archivo de configuración (zebra-core.config). El diseño altamente modular de la herramienta busca dar flexibilidad suficiente al proceso de modo que cada etapa del proceso pueda ser controlada independientemente desde una interface de alto nivel.

Se describen a continuación algunas de las características específicas de la herramienta y la manera como se implementa en ella los métodos y estrategias descritos en secciones precedentes.

4.1. Zebra-core-strips: Particionado

Este módulo realiza la tarea básica de particionado de la escena. El módulo recibe como parámetros de entrada la dirección del particionado por franjas (horizontal o vertical), el archivo que contiene la descripción de la escena en el lenguaje propio de POV-Ray (archivo .pov), el ancho y alto en pixels (W, H) y el número de franjas N en los que se desea particionar la imagen. Usando esta información el módulo calcula

las coordenadas de las esquinas de cada una de las franjas mediante la siguiente prescripción matemática:

$$(x_l, y_l)_i = ([i-1](W/N)+1, 1)$$
$$(x_r, y_r)_i = (x_l + W/N, H)$$

Esto para el caso de franjas verticales, Dónde l, r hacen referencia a esquina izquierda y esquina derecha respectivamente y el índice i toma valores entre 1 y N. Una fórmula análoga se aplica en el caso de franjas horizontales.

Una vez se han calculado las coordenadas de las franjas estas se almacenan en un archivo (strips.xy) que alimentará las etapas posteriores del proceso. Opcionalmente es posible indicar a través de un archivo .xy previamente preparado por el usuario las coordenadas de las franjas en las que se particionará la imagen. Esta característica permite al usuario (o a una versión posterior del software) realizar un particionado más equilibrado de la imagen teniendo en cuenta sus particularidades o las de la plataforma (balanceo de carga).

Con las coordenadas de cada franja el módulo genera un conjunto de scripts en bash que contienen las instrucciones requeridas para el rendering con POV-Ray de la franja. Estos scripts contienen además el código necesario para almacenar la salida estándar del motor de rendering y para registrar el comienzo y finalización de las tareas de rendering, informaciones que son de central importancia para el proceso de monitoreo y de evaluación de los resultados. El archivo con las coordenadas de las franjas y los scripts de renderización se almacenan en un directorio único para cada proceso (run).

4.2. Zebra-core-procs: aglomeración

Zebra-core-procs es el módulo que realiza la tarea de aglomeración. Como parámetros básicos se debe indicar el nombre del proceso de rendering (run) sobre el que se quiere realizar la tarea de aglomeración, el número de procesos final (que será también el número de jobs lanzados por el Scheduler si es el caso), el tipo de procesamiento (secuencial o distribuido) y una opción que permite que las subtareas en cada trabajo se lancen simultáneamente en el recurso donde el trabajo se ejecuta (esto último tiene el propósito de explotar el potencial que dicho recurso puede tener para la ejecución paralela de tareas secuenciales – múltiple procesador o tecnología Hyperthreading).

El proceso de aglomeración tiene como resultado un conjunto de scripts escritos en un lenguaje apropiado para el Job Scheduler que se utilice en la plataforma. Cada uno de esos scripts invoca la ejecución de un subconjunto de los scripts que generan las franjas y que fueron creados en la etapa de particionamiento por zebra-core-strips.

Zebra-core-launch: asignación

Una vez creados los scripts de lanzamiento de los trabajos en la etapa de aglomeración, el sistema esta listo para asignar las tareas a los recursos en la plataforma. Esta es la tarea del módulo zebra-core-launch. Cuando es invocado ese módulo prepara el espacio de almacenamiento de los archivos de salida e invoca o lanza uno a uno los scripts correspondientes a cada trabajo.

Cuando el trabajo de rendering se ha configurado como del tipo secuencial, los scripts de cada trabajo son lanzados usando el sistema de colas at. Este tipo de ejecución se realiza con propósitos comparativos y para la determinación de métricas como el speed-up y la eficiencia.

La configuración natural (y también por defecto) de un trabajo de rendering con ZeBrA-Core es aquella en la que se recurre a la ejecución distribuida. En este caso zebra-core-launch contacta al scheduler configurado en la plataforma y le entrega los scripts de lanzamiento de cada uno de los trabajos. Para la primera versión de la herramienta se asume la disponibilidad de un sistema de colas del Sun Grid Engine, una herramienta libre de gran difusión en sistemas de procesamiento distribuido. Naturalmente para versiones posteriores de la herramienta se espera ampliar la compatibilidad de la herramienta a una más amplia gama de schedulers.

4.3. Zebra-core-stat: monitoreo y síntesis de resultados

En todo sistema de cómputo distribuido el seguimiento y monitoreo de los procesos representa una de las más importantes tareas. Al no tener acceso inmediato a la salida estándar de los programas, los sistemas de monitoreo son mecanismos expeditos para conocer el estado de una tarea de procesamiento. ZeBrA-Core incluye a través de este módulo su propia herramienta de monitoreo que entrega reportes del estado de ejecución de los procesos y los resultados del rendering de las franjas. Zebra-core-stat además genera reportes finales y estadísticas del proceso de rendering que son de utilidad para la evaluación del mismo.

Las estadísticas más importantes generadas por este módulo en la primera versión de la herramienta incluyen:

- Tiempo de rendering por franja (t_i). Este valor se obtiene al comparar el tiempo en el que comienza efectivamente a ejecutarse el rendering de la franja y el tiempo en el que termina. No se incluyen en el los tiempos de espera y transferencia propios de la herramienta de asignación de trabajos.
- Media aritmética de los tiempos de rendering por franjas (t_m).
- Dispersión de los tiempo de rendering por franjas

(s_m). Esta métrica ofrece una medida del "desbalance" de carga obtenido en el proceso de particionado y aglomeración.
- Cociente de balanceo (B). Esta métrica permite obtener un indicativo del balance de carga en el rendering de las franjas. Se define como el cociente entre la dispersión de los tiempos de rendering y la media de esos mismos tiempos, B = sm/tm. En una tarea de rendering con un balanceo de carga ideal o de una imagen completamente homogénea B = 0. El valor de B se incrementa a medida que el proceso de rendering es menos balanceado entre franjas y recursos.
- Tiempo total de rendering distribuido (T). Esta es la métrica central del proceso. En su versión más sencilla T se calcula con la diferencia entre el momento de finalización del rendering de la última franja y el tiempo de inicialización de la primera (última y primera hacen referencia al momento de rendering no a la posición de las franjas). Un problema surge cuando los trabajos son aceptados en distintos momentos por el programador (por ejemplo en un ambiente de cómputo muy congestionado). En ese caso T se calcula como el mayor de los tiempos invertidos por los trabajos individuales.
- Tiempo acumulado (T_a). Igual a la suma directa de los tiempos de rendering de todas las franjas. Ofrece la medida de referencia de lo que tomaría completar el proceso de rendering en caso de que se ejecutará de forma secuencial.
- Speed-up (S_p). Definido como $Sp = T_a / T$.
- Eficiencia (E). Definida como el cociente entre el número de procesos y el speed-up, $E = N_p/S_p$.

4.4. Zebra-core-build: fusión

La etapa final del proceso de renderización es realizada por el módulo zebra-core-build. La construcción final de la imagen es realizada por este módulo en tres pasos: primero transfiere las imágenes de las franjas individuales desde el sistema de archivos en las que son colocadas directamente por POV-Ray (denominado en el contexto de la herramienta el scracth directory) hasta el sistema de archivos local. Esta transferencia puede implicar el uso de la red, si el sistema de archivos fue definido como local en el recurso que realizo el rendering de cada franja o simplemente involucrar una copia desde un sistema de archivos local, cuando se utilizan sistemas de archivos distribuidos (NFS, pvfs, etc.)

En segunda instancia las imágenes generadas son convertidas a un formato binario más compacto que el utilizado por POV-Ray (por defecto la primera versión de ZeBrA-Core usa el formato Targa no comprimido). Se usa PNG como formato final de las porciones de imagen. En tercer y último lugar las imágenes son fusionadas en una sola, para lo cual ZeBrA-Core recurre al módulo 'montage' de la librería ImageMagick [13]. Al terminar su tarea zebra-core-build entrega una imagen completa en formato PNG de la escena provista por el usuario.

5. Pruebas y resultados

Con el objeto de poner a prueba la efectividad de las técnicas y métodos para el rendering distribuido presentadas en este trabajo y para ofrecer una plataforma completa de prueba para ZeBrA-Core se llevaron a cabo una serie de tareas de rendering de algunas escenas muy populares en la comunidad de POV-Ray. Las pruebas se realizaron explorando exhaustivamente el espacio de parámetros del problema.

Los parámetros considerados en las pruebas fueron los siguientes:

- Escena (e) - e_1: chess2, e_2: skyvase
- Resolución (r) - r_1: 1440x1080, r_2: 4320x3240, r_3: 8640x6480
- Dirección de particionado (d) – d_1: horizontal, d_2: vertical
- Tipo de procesamiento (p) – p_1: distribuido (ZeBrA-Core), p_2: paralelo (MPI-POVRay)
- Número de franjas (N) – N_1: 1, N_2: 2, N_3: 5, N_4: 10, N_5: 15, N_6: 20

Para cada prueba y dada la disponibilidad de recursos se realizaron corridas usando un número de trabajos (número de procesadores) igual al de franjas (Np = N). Se realizaron en total cerca de 120 pruebas que tomaron conjuntamente un total de más de 80 horas de procesador. Las pruebas incluyeron la comparación de procesos de rendering distribuido usando la herramienta presentada en este trabajo y una herramienta paralela, MPI-POVRay [3].

Todas las pruebas se corrieron sobre el Cluster "Hércules" del Instituto de Física de la Universidad de Antioquia, un Cluster Beowulf "homogéneo" con 22 procesadores, 2 GB/procesador y una red de 1 Gbit/s.

Con el objeto de analizar los resultados obtenidos en las tareas de rendering se formularon las siguientes preguntas:

1. ¿Cómo se reduce el tiempo de procesamiento con el número de franjas? ¿disminuye según lo esperado?
2. ¿Cuál es la eficiencia en la utilización de recursos?
3. ¿Cómo se compara el tiempo de procesamiento cuando se hace el rendering con franjas verticales y con franjas horizontales?
4. ¿Cómo se compara el tiempo de procesamiento usando ZeBrA-Core y una herramienta para rendering en paralelo?

5. ¿Cómo varía el speed-up y eficiencia del rendering con la complejidad y resolución de la imagen?

Para responder estas preguntas se construyeron a partir de los resultados los gráficos presentados en las figuras 4 a 9. En estos gráficos se estudia el comportamiento de distintas métricas en función del número de franjas (número de procesadores) dejando constantes otros parámetros libres de la prueba.

6. Discusión y conclusiones

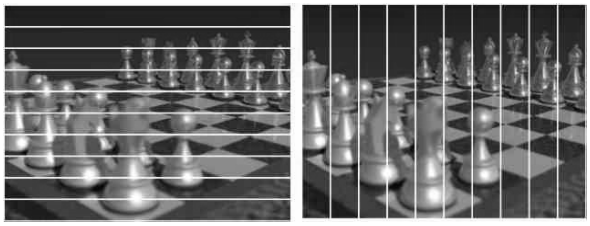
Se presento en este trabajo una nueva herramienta para la generación de imágenes fotorrealistas usando POV-Ray en una plataforma de computación distribuida. La herramienta, en su primera versión utiliza un esquema de particionado simple (particionado por franjas) y una estrategia de distribución de procesos basada en la utilización de Job Schedulers. Por su generalidad la herramienta puede utilizarse para realizar tareas de rendering distribuido sobre plataformas muy diversas y usando cualquier versión de POV-Ray. Esto último aunado a una baja tolerancia a las fallas, una característica que esta implícita por la estrategia de distribución utilizada, le da ventajas a la herramienta respecto a otras que recurren al cálculo en paralelo (MPI-POVRay y PVM-POVRay).

La herramienta, los métodos y la estrategia de distribución utilizada fueron puestas a prueba en un conjunto diverso de condiciones. Como era de esperarse el tiempo de rendering disminuyó al aumentar el número de franjas en las que se dividieron las imágenes (figura 4). El speed-up mostró un buen comportamiento especialmente para la escena chess2 (figura 5), que es también la más compleja de las dos escenas utilizadas. Aún para un número relativamente grande de procesos (20) el speed-up en este último caso alcanzo un nivel de 15. El caso más extremo de reducción en el tiempo de ejecución se obtuvo para la escena chess2 a una resolución de 8640x6480. Mientras que la generación en secuencial de esta imagen toma 10 horas 20 minutos, usando 20 procesos el rendering toma solamente 37 minutos.

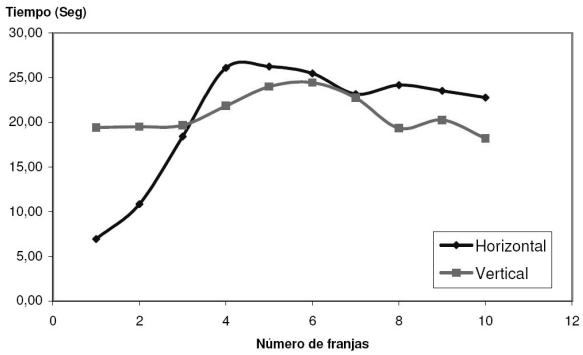
La eficiencia en el uso de los recursos de cómputo en la plataforma es siempre muy alta (más de un 80% para las resoluciones más altas) un buen indicador de la bondad del método en relación con su capacidad para explotar a fondo los recursos disponibles (figura 6). Uno de los resultados más curiosos fue la observación sistemática de que el tiempo de rendering de la imagen chess2 cuando se usan bandas horizontales es siempre mayor que el requerido cuando se usan bandas verticales (a igual número de bandas) (figura 7). La explicación de este comportamiento se obtiene estudiando el tiempo de rendering individual de cada franja en ambos casos (figura 10) y las métricas descritas en 4.4. Como puede apreciarse el rendering horizontal es mas desbalanceado que el vertical, esto debido a que las primeras franjas en el primer caso incluyen una región del cielo muy homogénea y rápida para renderizar (figura 2). El resultado es que aunque más complejas en promedio las tareas de las franjas verticales más equilibrada es la carga que se entrega a cada procesador. Esto es un significativo indicador de la necesidad de algoritmos de balanceo de carga con los que se pueden obtener, según se puede estimar de la figura 7 ganancias de hasta el 15% en el tiempo de rendering.

En condiciones similares el rendering con ZeBrA-Core produjo resultados mejores que el rendering usando procesamiento paralelo (figura 8). Este es un indicativo muy positivo en favor de los métodos y las estrategias propuestas en este trabajo, en tanto las herramientas paralelas son de las más utilizadas para el rendering en plataformas distribuidas. Pero el rendering distribuido no es tampoco la panacea en todas las condiciones. Su capacidad para disminuir consistentemente el tiempo de generación y de utilizar eficientemente la plataforma, se deteriora cuando la imagen es de una resolución baja (como se aprecia en las figuras 6 y 9) o cuando no es muy compleja como se observa de la comparación del speed-up en el caso de la escena skyvase y chess2 en la figura 5.

El trabajo presentado aquí deja abiertas posibilidades para el desarrollo posterior de esta y otras herramienta para rendering distribuido, alternativas de las ya clásicas herramientas de rendering paralelo desarrolladas en la última década y media. A bajo nivel (en el que se encuentra actualmente la herramienta) es de interés desarrollar e implementar algoritmos para balancear mejor el rendering de cada franja, un esfuerzo que podría impactar positivamente los tiempos de generación. Se hace necesario también expandir la compatibilidad de la herramienta con otros Schedulers comunes (PBS, Condor, Maui, LSF). Estudiar su utilización sobre el sistema operativo Windows y su compatibilidad con Schedulers desarrollados para este sistema (Alchemi por ejemplo). A alto nivel el paso inmediato es desarrollar una interface gráfica que permita a un usuario menos avezado en el uso de scripts y de la línea de comandos realizar tareas complejas de rendering. Así mismo la flexibilidad de ZeBrA-Core es suficiente para desarrollar una interface Web que permitiera ofrecer a usuarios selectos el servicio de rendering distribuido. Es posible concebir también la posibilidad de utilizar la herramienta para montar y operar una granja de rendering que permitiera vender servicios de generación de imágenes a usuarios en sectores como el diseño gráfico e industrial, la finca raíz, museos, empresas de desarrollo de software, animación 3D y cine animado.



a)



b)

Figura 10. *a) franjas resultantes de la división horizontal y vertical de la escena chess2. b) Tiempo de rendering individual para 10 franjas en la escena chess2. Las métricas en ambos casos son: horizontal – t_m= 20.7, s_m= 6.7, B= 0.32 ; vertical – t_m= 20.9, s_m= 2.1, B =0.1*

7. Agradecimientos

Los autores agradecen al Instituto de Física por permitir el uso del Cluster Hércules para el desarrollo de la herramienta y la ejecución de las pruebas. Agradecemos también las observaciones y sugerencias útiles de los profesores Andrés Marín y Francisco Guevara. Este trabajo fue motivado por los ejercicios realizados por dos de los autores (Escobar y Bedoya) en el marco del Seminario de Procesamiento en Paralelo que orientaron los profesores Marín, Guevara y Zuluaga en el Departamento de Sistemas durante el segundo semestre del 2006 y el primer semestre de 2007.

8. Referencias

[1] Bedoya D., Escobar C., Zuluaga J. En preparación. 2007.

[2] Christensen P., Fong J., Laur D., Batali D. "Ray Tracing for the Movie 'Cars'". Proceedings of the IEEE Symposium on Interactive Ray Tracing. 2006.

[3] Fava A., Fava E., Bertozzi M. "MPIPOV: a Parallel Implementation of POV-Ray Based on MPI", European PVM/MPI Users' Group Meetings. 1999.

[4] Foley J., Van Dam A., Feiner S., Hughes J. "Interactive Computer Graphics: Principles and Practice". Addison-Wesley, 1995.

[5] Foster I. "Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering". Addison Wesley, 1995.

[6] Freisleben B., Hartmann D., Kielmann T. "Parallel Ray tracing: A Case Study on Partitioning and Scheduling on Workstation Clusters", 30th Hawaii International Conference on System Sciences. 1997.

[7] Heirich A, Arvo J. "Scalable Monte Carlo image synthesis". Elsevier Science Publishers B. V. 1997.

[8] Magnenat-Thalmann N., Pandziz I., Moussalay J., Thalman D., Huang Z., Shen J. "The making of Xian Terra-Cotta Soldiers," Computer Graphics: Developments in Virtual Environments, p.p. 281-295, Academic Press, 1995.

[9] Persistence of Vision Raytracer Pty. Ltd. "POV-Ray 3.6.1 Documentation". Documentation of POV-Ray. 2007.

[10] Plachetka T. "Persistence Of Vision Parallel Raytracer", 14th Spring Conference on Computer Graphics. 1998.

[11] Plachetka T. "Tuning of Algorithms for Independent Task Placement in the Context of Demand-Driven Parallel Ray Tracing," Eurographics Symposium on Parallel Graphics and Visualization. 2004.

[12] Segal M., Akeley K. "The OpenGL Graphics System". Documentation of OpenGL. 1997.

[13] Still M. "The Definitive Guide to ImageMagick ". Apress. 2005

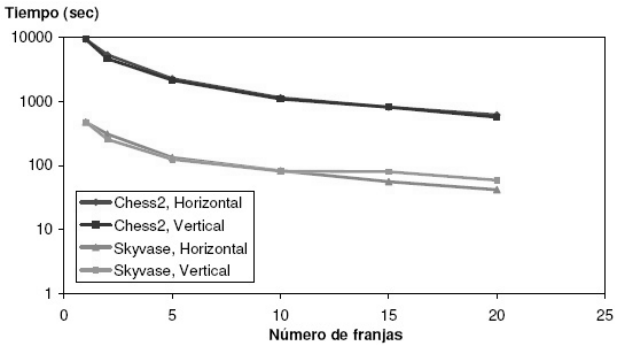


Figura 4. Tiempo de procesamiento (p_1, r_2)

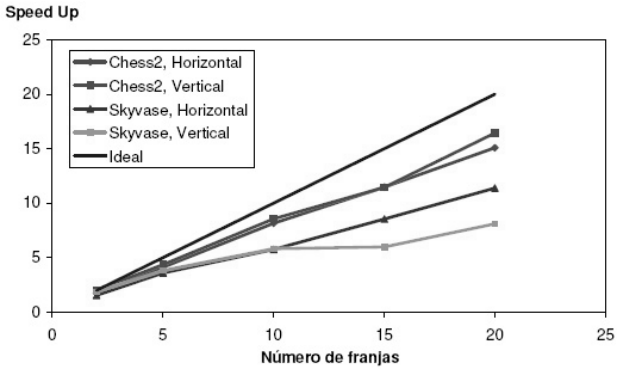


Figura 5. Speed-up (p_1, r_2)

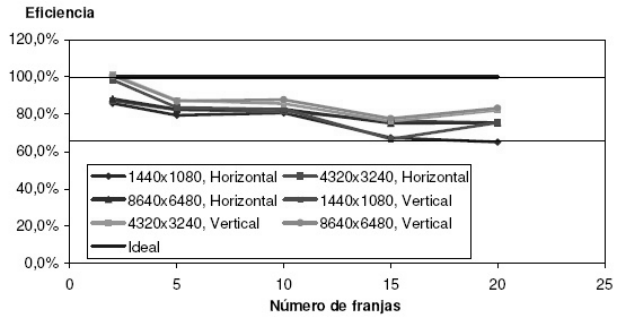


Figura 6. Eficiencia (i_1, p_1)

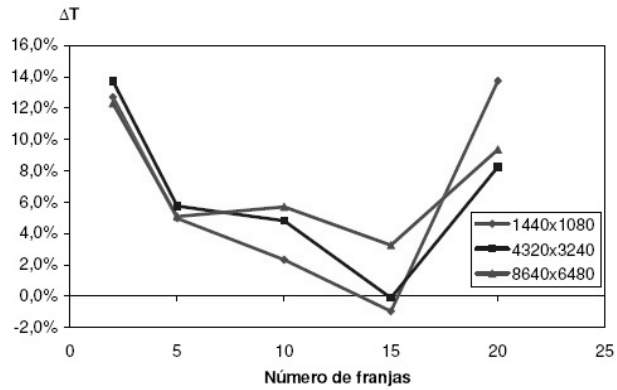


Figura 7. Comparación del tiempo de rendering cuando se particiona la imagen en franjas verticales y horizontales. DT/ T = [T(d_1) – T(d_2)] / T(d_1). Un valor positivo indica un tiempo mayor de procesamiento con franjas horizontales (i_1, d_1, p_1)

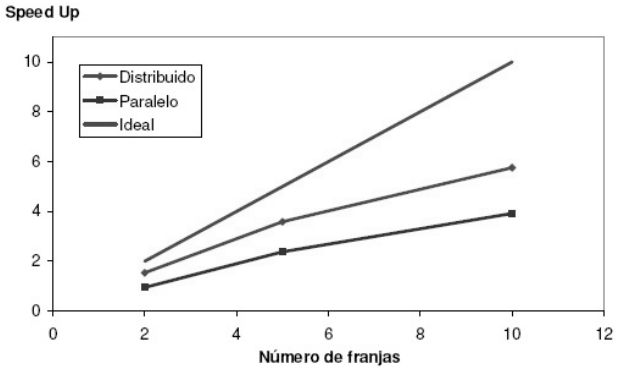


Figura 8. Comparación del Speed-up cuando se usa procesamiento distribuido y en paralelo (i_2, d_2)

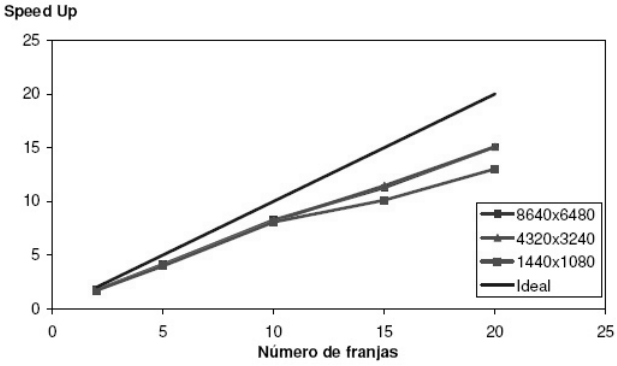


Figura 9. Comparación del speed-up para distintas resoluciones de la misma imagen (i_1, d_1, p_1)