Evaluación de una solución de renderizado distribuido en las salas de cómputo de la Universidad Icesi para los estudiantes de Diseño Industrial y de Medios Interactivos

Evaluation of a distributed rendering solution in the computer room of Universidad Icesi for the Industrial Design and Interactive Media Design students.

Juan David Osorio Betancur, Ing.
Grupo i2T, Universidad Icesi, Colombia
juandoso@gmail.com

Fecha de recepción: 08-02-2010 Fecha de selección: 30-03-2010 Fecha de aceptación: 15-03-2010

ABSTRACT

This article presents the results of an evaluation made in the Universidad Icesi to find solutions to the bottlenecks and long rendering times affecting the Design students while working with commercial design packages during their courses.

In particular, commercial render farm managers are evaluated, and a proof-of-concept render farm manager based in open source opportunistic computation software is developed.

KEY WORDS

Distributed rendering, render farms.

RESUMEN

En este artículo se presentan los resultados de una evaluación realizada en la Universidad Icesi para encontrar alternativas de solución al problema de cuellos de botella y largos tiempos de renderizado que sufren los estudiantes de los programas de Diseño al trabajar con paquetes de diseño en sus cursos. En particular, se evalúan alternativas comerciales de administradores de granjas de renderizado, y se desarrolla una prueba de concepto de un administrador de granja de renderizado basado en paquetes open source de computación oportunista.

PALABRAS CLAVE

Renderizado distribuido, granjas de renderizado.

Clasificación Colciencias: Tipo 5

1. INTRODUCCIÓN Y MOTIVACIÓN

El renderizado es el proceso de generar una imagen desde un modelo. El modelo es una descripción de un objeto o escena tridimensional en un lenguaje o estructura de datos estrictamente definido, que contiene información sobre geometría, iluminación, texturas y puntos de vista. Este es un proceso computacionalmente complejo, por lo que el renderizado de un modelo complejo puede tardar de horas a días, incluso en una estación de trabajo moderadamente poderosa.

Particularmente en el caso de la Universidad Icesi, los estudiantes de diseño industrial y diseño de medios interactivos crean regularmente trabajos (en programas como Solid Works, Maya, Adobe premiere, After effects, 3D Studio Max) que tardan del orden de veinticuatro horas en ser renderizados en PCs de las salas de cómputo, siendo este un tiempo en el que es prácticamente imposible utilizar el computador para otras actividades, por la enorme cantidad de recursos que esta actividad exige. Más aun, si el estudiante comete un error en el modelo o desea hacerle cambios al producto final, todo ese tiempo de renderizado se pierde y es necesario empezar de nuevo.

Por otro lado, el hecho de que las licencias de los programas de diseño tienden a ser costosas hace necesario limitar el número de equipos donde están instaladas, lo que unido al creciente número de estudiantes de estas carreras hace que el tiempo de uso de los PCs de las salas de diseño sea cada vez un recurso más crítico.

Es por esto que encontrar una solución óptima de renderizado que reduzca significativamente el tiempo que un estudiante debe dedicar a esta tarea es vital para eliminar este cuello de botella.

La renderización distribuida es un tema razonablemente maduro, en el sentido que existen diversas herramientas de software comerciales, algunas de las cuales son ampliamente utilizadas por estudios de diseño profesionales, que permiten instalar una solución más o menos 'out of the box'. Usualmente estas herramientas (siendo las más conocidas RenderPal y Smedge) están enfocadas en tener una granja de servidores dedicados a la tarea, y se licencian de acuerdo con el número 'engines' de renderizado, es decir, al número de servidores de la granja.

Además, hay algunas herramientas open source (como DrQueue) que siguen esta misma aproximación, pero que requieren dedicarle más trabajo al tema de la compilación desde la fuente, instalación y configuración.

En este artículo se hace una descripción del proceso de selección y pruebas de una herramienta de renderizado distribuido para descargar los computadores individuales de este proceso y optimizar el uso de recursos de hardware y software en salas. Se hace una breve exploración al uso de herramientas de dominio público y se exploran alternativas para este proceso. El artículo está distribuido de la siguiente forma: En la sección 2 se hace la descripción del proceso de implementación y la arquitectura seleccionada; en la sección 3 se muestran los resultados obtenidos, en la sección 4 se revisa brevemente el tema de renderizado oportunista y en la sección 5 las conclusiones y trabajo futuro.

2. DESCRIPCIÓN DE LA IMPLEMENTACIÓN

2.1. Render Farm Managers, y RenderPal

La solución típica para el problema de los tiempos de renderizado en una organización es utilizar una infraestructura de granjas de renderizado (render farm) y un software de administración de las mismas (render farm manager).

Una render farm es simplemente un grupo de estaciones de trabajo o servidores de altas capacidades dedicados a la única función de realizar trabajos de rendering on-demand. Estas máquinas usualmente estarán ubicadas en un cuarto de servidores donde su temperatura de funcionamiento será controlada (por ser máquinas de altas prestaciones), tendrán suministro eléctrico ininterrumpido (por medio de UPS) y estarán enlazadas a dispositivos de conectividad de alta velocidad (con el fin de reducir los tiempos de transferencia de los archivos de renderizado, que usualmente son grandes). También es usual tener un sistema de almacenamiento centralizado de los proyectos de renderizado y sus resultados, con altas capacidades de almacenamiento y conectado a la red por medio de un vínculo de alta velocidad. Es importante tener en cuenta los temas de licenciamiento de los motores de renderizado que se utilicen. Cada una de las estaciones que conforman la render farm debe tener activa una licencia del software utilizado para realizar el renderizado (o más de una, en caso que el software en cuestión se licencie por procesador), y dado que usualmente estas licencias son costosas es un punto muy importante al momento de hacer el presupuesto. Por ejemplo, lo usual para una organización grande es comprar licencias concurrentes para sus paquetes de diseño, pues esta es probablemente la forma más eficiente de manejar las licencias donde el número de usuarios del software en un momento determinado es menor que el número de máquinas donde está instalado. Sin embargo si se usa este esquema cada una de las estaciones de renderizado acaparará una licencia todo el tiempo, dado que lo ideal es que el tamaño de la granja se dimensione para que todas las máquinas estén trabajando todo el tiempo para así aumentar el retorno sobre la inversión, y obviamente el resultado será que los usuarios tendrán disponibles menos licencias de las esperadas.

Un render farm manager es un paquete de software que se encarga de las labores de monitoreo y administración de la render farm y los trabajos de renderizado, realizando funciones como creación de trabajos (usualmente desde un plugin instalado en la aplicación de diseño utilizada), monitoreo de los trabajos en progreso, notificación de trabajos terminados por email u otro medio, balanceo de carga entre las estaciones de renderizado, y otras.

La estructura de esta solución se muestra en la Figura 1.

Los render farm managers más utilizados son Smedge[1] (de Überware) y RenderPal[2] (de Shoran Software). Las características de ambos son muy similares:

- Configuración de parámetros de renderizado: Aunque lo normal es que los trabajos de renderi-

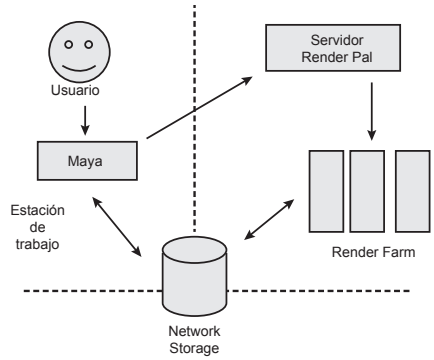

Figura 1. Arquitectura de una solución típica de distribución de renderizado basada en granja de servidores

zado enviados a una granja se procesen de acuerdo con los parámetros configurados desde el programa de diseño, también es posible modificar estos parámetros en cualquier momento desde la interfaz de administración del Manager.

- Monitoreo de trabajos y máquinas: Es posible monitorear el estado de los trabajos en cola o en proceso, así como el estado de las estaciones de trabajo que componen la granja.
- Control sobre los trabajos: Los trabajos pueden priorizarse, pausarse o cancelarse en cualquier momento.
- Soporte de paquetes de diseño: Integración con los paquetes de diseño más utilizados en el mercado, entre otros Maya, 3ds Max, After Effects, Cinema 4D, etc.
- Extensibilidad: Es posible extender el soporte del render farm manager hacia otros paquetes de diseño no soportados, por medio de lenguajes de scripting (p. ej. Python) siempre y cuando el paquete en cuestión pueda ser accedido por medio de línea de comandos.
- Distribución de trabajos de renderizado: Cada trabajo de renderizado puede dividirse para que sea procesado por más de una estación de trabajo simultáneamente. Esta división puede hacerse de acuerdo con varios métodos (que se configuran al momento de crear el trabajo de renderizado): frame splitting, image slicing, capas individuales de renderizado, múltiples cámaras, etc.
- Descubrimiento automático de clientes y estaciones: Para facilitar la administración de la granja, el administrador detecta automáticamente las estaciones de trabajo donde se ha instalado su aplicación cliente.
- Administración automática de discos de red y sus rutas: Dado que todas las máquinas del sistema deben poder acceder a un sistema de almacenamiento de red, es necesario administrar las rutas por medio de las cuales se accede a este sistema. Si esta administración es automática se eliminan posibles fuentes de error y downtimes de la granja.

RenderPal, sin embargo, destaca en algunas de sus características:

- Wake-on-LAN: Las estaciones de renderizado pueden permanecer apagadas hasta que se necesiten, reduciendo el consumo de energía. (Smedge lo planea para su versión 2011).

- Actualizaciones automáticas: Todos los componentes del sistema pueden actualizarse automáticamente y en funcionamiento (Smedge lo planea para su versión 2011).
- Integración con VNC: Las estaciones de renderizado pueden accederse remotamente usando VNC, en forma integrada con la aplicación de administración.

Además, RenderPal destaca por dos características especialmente importantes para el uso en la Universidad: Su precio (la licencia académica para volúmenes bajos cuesta EU$25 por estación, mientras que la de Smedge cuesta US$50), y su integración con Maya, puesto que la integración con el paquete de diseño es fácil de instalar y el plugin es supremamente sencillo de manejar, lo que es muy importante puesto que los usuarios típicos son estudiantes.

Adicional a los render farm managers comerciales hay una alternativa open source muy madura y muy utilizada en el mercado llamada DrQueue,[3] que realiza prácticamente todas las funciones de los programas comerciales antes mencionados. Sin embargo, tiene dos debilidades importantes: La primera es que su distribución se da exclusivamente a nivel de fuente (i. e. no existe un instalador estándar) y el proceso de compilación desde la fuente es complejo y propenso a errores, y la segunda es la ausencia de un plugin para su integración directa con los paquetes de diseño más utilizados (están en desarrollo plugins para Blender y K3D, que son paquetes de diseño open source). Así, aunque es posible utilizar DrQueue como render farm manager para renderizar trabajos de Maya, por ejemplo, la creación de estos trabajos no se da desde la interfaz misma de Maya (como sí ocurre con RenderPal) sino que es necesario realizarla desde la interfaz de línea de comandos de DrQueue, o desde una interfaz web adicional que no hace parte de la distribución estándar del programa.

Las pruebas de estos productos se realizaron con máquinas virtuales para instalar las versiones de prueba de los servidores de los distribuidores de renderizado, una workstation Dell (Intel Core2 Quad Q9300 a 2.50GHz, 4 GB de RAM, tarjeta de video Nvidia Quadro FX570) como estación de renderizado, y un computador de escritorio como cliente. Se instalaron las versiones de prueba de los distribuidores y se hicieron pruebas con dos proyectos típicos en Maya de los cursos de diseño de la Universidad. Aparte de las diferencias mencionadas arriba, el renderizado como tal da los mismos resultados, sin importar qué distribuidor se utilice, ya que el distribuidor se encarga solo de delegar el proceso de renderizado, del cual se ocupa el motor (engine) del paquete de diseño a utilizar.

Pasar de una configuración de pruebas como esta a una definitiva es relativamente sencillo, desde el punto de vista del software, una vez se tienen las licencias, pues los distribuidores de renderizado pueden detectar cuando en una nueva máquina se ha instalado el programa cliente, y desde la interfaz de administración se puede añadir esta máquina a la granja de renderizado con la configuración que se quiera. Así, tener una granja de renderizado en producción se limita a instalar el cliente del distribuidor

en un conjunto de estaciones de trabajo dedicadas. Sin embargo, desde el punto de vista de hardware esta propuesta no es tan simple, pues hay muchas otras consideraciones a tener en cuenta, como el espacio de rack a utilizar en el cuarto de servidores, la carga adicional en cuanto a enfriamiento y consumo de energía, y la necesidad de contar con conexiones de alta velocidad entre los desktop de los clientes, la unidad de almacenamiento y las estaciones de renderizado para mover archivos de proyectos relativamente grandes.

3. ANÁLISIS DE TIEMPOS Y PROYECCIONES

Para ilustrar el impacto que tiene el número de núcleos de procesamiento disponibles sobre los tiempos de renderizado de un trabajo típico, se hicieron pruebas con los dos proyectos de ejemplo en la workstation, se dejó disponible al proceso de uno a cuatro núcleos de la CPU, y se usó una opción de configuración de RenderPal. Los resultados se muestran a continuación.

Como regla general, a medida que aumenta el número de núcleos disponibles para las operaciones de renderizado, el tiempo total disminuye en forma asintótica, como puede verse en las Figuras 2 y 3.

Hay que tener en cuenta al hacer proyecciones que si aumenta el número de núcleos disponibles, pero estos se encuentran ubicados en otras máquinas conectadas en red, se incurre necesariamente en cierto overhead que si bien puede no ser significativo para el caso de un trabajo de renderizado que tarde cientos de minutos, sí puede afectar el desempeño si se trata de decenas de trabajos que tarden decenas de minutos cada uno.

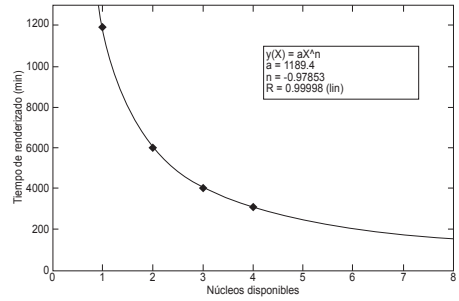

Figura 2. Tiempo de renderizado en función de los núcleos de procesamiento utilizados.

Puntos y proyección, Escena 1.

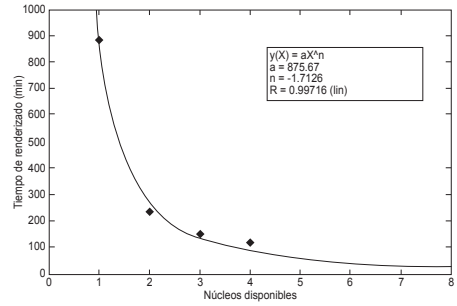

Figura 3. Tiempo de renderizado en función de los núcleos de procesamiento utilizados.

Puntos y proyección, Escena 2.

En las imágenes se grafican los tiempos de renderizado de dos escenas, dejando disponible para cada una de uno a cuatro núcleos de un equipo quad-core, junto con la curva que mejor se ajusta a los puntos (obtenida con el toolbox Ezyfit para Matlab).[4]

En las gráficas se puede apreciar una reducción exponencial en el tiempo de renderizado, según la cantidad de núcleos disponibles. Estas gráficas tienen una asíntota respecto al número de núcleos, lo que implica que es necesario buscar un punto óptimo

en términos del número de núcleos de rendererizado y el costo de la solución. Estos resultados son mucho más marcados para la escena 1.

Los tiempos de renderizado y los parámetros de la curva de ajuste de la forma $y = a\,x\,\hat{}\,n$, pueden verse en las Tablas 1 y 2, respectivamente.

Tabla 1: Tiempos de renderizado en minutos

Núcleos	Escena 1	Escena 2
1	1190	880
2	602	236
3	404	147
4	310	115

Tabla 2: Parámetros de la curva de ajuste

	Escena 1	Escena 2
A	1189.4	875.7
N	-0.9785	-1.7126
R	0.99998	0.99716

4. RENDERIZADO USANDO COMPUTACIÓN OPORTUNISTA: BOINC-PY-RENDER

Por otro lado, hay otro enfoque al concepto de renderización distribuida, y consiste no en tener equipos dedicados 100% del tiempo a la tarea, como en una granja de servidores, sino en aprovechar el tiempo de cómputo desperdiciado en equipos no dedicados conectados a la red: los computadores de las salas de sistemas y del área administrativa de la Universidad, por ejemplo. Este esquema de trabajo se conoce como Desktop Grid Computing o computación oportunista, y aunque su aplicación al problema de renderizado es una línea de trabajo que puede ser muy promisoria, las herramientas que siguen este paradigma son aún trabajo en curso y están limitadas al uso de paquetes de diseño 3D open source, como Blender y Yafray.

Como parte del proyecto, se diseñó una prueba de concepto de distribuidor de renderizado basado en BOINC (un middleware para Desktop Grid Computing), denominada boinc-py-render, que incluye como componentes un sitio web basado en Django[5] como interfaz de usuario, una librería de Python para computación en paralelo basada en el paradigma master-worker llamada PyMW[6,7] para el control de los trabajos de renderizado, y BOINC[8,9] como middleware de comunicación con los nodos (Figura 4).

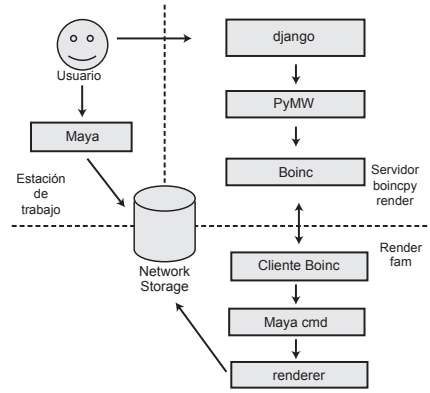

Figura 4. Arquitectura de la prueba de concepto desarrollada

El funcionamiento es como sigue:

1. El usuario realiza su diseño y almacena su proyecto en un disco de red, desde donde será accesible por la estación de renderizado.
2. El usuario ingresa al sitio web de boinc-py-render, y allí creará un nuevo trabajo de renderizado, anotando un nombre del trabajo y la ruta a la carpeta del proyecto en el disco de red.

3. Una vez ingresado el nuevo trabajo, PyMW se encarga de crear una nueva work unit (que es la unidad básica de trabajo distribuido de BOINC), y la almacena en la base de datos del servidor BOINC. Esta work unit contiene solo la meta-información del trabajo de renderizado (identificación del trabajo, y comando de mayabatch a ejecutar en la estación), puesto que toda la información necesaria para el propio renderizado se halla en la carpeta compartida del proyecto.
4. Cuando una estación de renderizado se encuentra inactiva, el cliente BOINC instalado se encarga de solicitar una nueva work unit al servidor.
5. Una vez recibida la work unit del servidor, la estación procede a desarrollarla, lo que consiste simplemente en ejecutar un comando de Maya batch que ordena al motor de renderizado realizar el trabajo descrito.
6. Cuando termina el proceso de renderizado, el resultado queda almacenado en la carpeta compartida del proyecto, y el cliente BOINC en la estación de trabajo informa del resultado al servidor BOINC, que almacena este informe en la base de datos. Hay que tener en cuenta que en este punto no hay notificación al usuario del término de su trabajo.
7. El usuario puede monitorear el estado de su trabajo a través de la interfaz web, que a su vez utiliza PyMW para obtener el estado de la work unit de BOINC. Este estado puede ser pendiente de envío, en proceso o terminado. Un trabajo terminado puede a su vez ser exitoso o fallido, en caso que haya habido algún error durante el renderizado.
8. Una vez el usuario observe que su trabajo terminó, puede acceder al resultado en la unidad de red que contiene la carpeta de su proyecto.

Esta solución cumple las funciones básicas de un distribuidor comercial (creación de trabajos de renderizado, envío de estos a estaciones de renderizado disponibles, notificación al sistema de trabajos terminados), pero no las funciones más avanzadas, que son el valor agregado de estos distribuidores comerciales. Sin embargo, muchas de estas funciones podrían desarrollarse e integrarse con la solución, aunque el costo en tiempo de desarrollo probablemente no justificaría este trabajo, puesto que diversas soluciones comerciales ya las implementan, han sido ampliamente probadas por el mercado, y lo hacen a un precio asequible. Como posibilidades de mejora vale la pena mencionar en primer lugar la integración con Maya, dado que esta utiliza Python como lenguaje de scripting para el desarrollo de plugins, sería relativamente fácil desarrollar uno para integrar directamente boincpy-render con la interfaz gráfica de Maya.

5. CONCLUSIONES Y TRABAJO FUTURO

El campo de las técnicas de aceleración de la renderización se ha vuelto extremadamente dinámico en los últimos años, debido en gran parte a los avances en el tema de la ren-

derización offline y el uso de tarjetas gráficas comerciales, tema que se abordará en un próximo artículo.

En cuanto a la renderización distribuida, el tema de las render farm es muy maduro y existen productos comerciales que pueden suplir adecuadamente las necesidades de la mayoría de las empresas. En la Universidad Icesi se espera implementar una solución como ésta a pequeña escala en el corto plazo, con el fin de aliviar las necesidades inmediatas de los estudiantes de los programas de Diseño.

Por otro lado, la renderización basada en computación oportunista es muy promisoria, e interesante como tema de investigación, pero aún requiere mucho trabajo para llegar a ser una alternativa viable.

BIBLIOGRAFÍA

1 http://www.uberware.net/smedge/features.php

2 http://www.renderpal.com/about.php

3 http://www.drqueue.org/cwebsite/about_drqueue.php

4 http://www.fast.u-psud.fr/ezyfit/

5 http://www.djangoproject.com/

6 Eric M. Heien, Yusuke Takata, Kenichi Hagihara, Adam Kornafeld, "PyMW - A Python module for desktop grid and volunteer computing," Parallel and Distributed Processing Symposium, International, pp. 1-7, 2009 IEEE International Symposium on Parallel & Distributed Processing, 2009

7 http://pymw.sourceforge.net/

8 David P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," Grid Computing, IEEE/ACM International Workshop on, pp. 4-10, Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), 2004

9 http://boinc.berkeley.edu/

CURRÍCULO

Juan David Osorio Betancur. Ingeniero Telemático de la Universidad Icesi. Investigador del grupo i2T con intereses en las áreas de computación de alto desempeño basada en desktop grid y tarjetas gráficas, sistemas distribuidos y aplicaciones móviles.