CEIG'09, San Sebastián, Sept. 9-11 (2009)
C. Andújar and J. LLuch (Editors)

Yafrid-NG: A Peer to peer Architecture for Physically Based Rendering

J. A. Mateos, C. Gonzalez-Morcillo, D. Vallejo and L. M. Lopez-Lopez

JoseAngel.Mateos@uclm.es, Carlos.Gonzalez@uclm.es, David.Vallejo@uclm.es, LorenzoManuel.Lopez@uclm.es
Escuela Superior de Informática - University of Castilla-La Mancha (Spain)

Abstract

Yafrid-NG is a peer to peer architecture and a set of protocols which reduces the time spent in the rendering phase by making use of a set of heterogeneous computers, distributed over the Internet, which will supply some of their resources for rendering 3D scenes. The system is completely decentralized and it takes advantage of p2p networks. The set of protocols needed for transferring files, the rendering process, and the results recovery and composition are also defined. Yafrid-NG is specifically designed for physically based rendering methods and the division of the work is optimized for that. We make use of a mechanism based on the scene properties to balance the complexity of the work units. Experimental results are presented to illustrate the benefits of using Yafrid-NG.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.2]: Distributed/network graphics—Computer Graphics [I.3.3]: Paralell processing—Computer Graphics [I.3.7]: Raytracing—

1. Introduction

Because rendering is a computationally intensive task, it is considered a bottleneck in graphics production. In order to address this issue, some works try to reduce the rendering time. Three development areas can be distinguished [CDR02]:

- Hardware Optimizations: Some developers use programmable GPUs (Graphics Processing Units) to improve the rendering process [BFH*04]. The main disadvantage is that this approach is specific for each hardware architecture.
- Improvement of rendering algorithms: Rendering algorithms are continuously being improved. A relevant approach consists of parallelizing the rendering algorithms to be executed in multiprocessor machines [SST06].
- Distributed Computing: Most of the rendering systems that use this approach are clustering or rendering farms that belong to an exclusive company. These systems are made up of homogeneous computers and all resources are managed in a centralized way. Another approach consists in using the idle time of a set of heterogeneous computers distributed over the internet. This is known as *Volunteer computing* [AF06]. One relevant approach to deal with high computational demands is the Grid [FK04].

This work defines a p2p architecture aimed at reducing rendering time. The system is composed of a set of heterogeneous machines connected to the Internet and each one of them will perform a portion of the whole work. We have based our work on the concepts of grid and *Volunteer Computing* to distribute the work among heterogeneous machines in terms of Hardware and Software. Grid particularities are completely transparent for the final user. The main advantages of this architecture are scalability, independence from the rendering method, and decentralization. It splits the tasks taking into account the complexity and the size of each work unit. Furthermore, it provides high performance with a high number of nodes, fine-grained (division of each frame into pieces) and coarse-grained (division of the work at frame level) granularity.

The rest of the paper is structured as follows. Section 2 provides an overview of related work previously developed in distributed rendering. Section 3 provides a general overview of the Yafrid-NG architecture. Section 4 describes the rendering process and each of the defined steps is explained specifically. Section 5 and Section 6 offer empirical results obtained with the proposed system, and discuss conclusions and future work guidelines, respectively.

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/227-230

2. Related Work

The number of approaches for non-interactive distributed photorealistic rendering is not very high. Dr Queue [DRQ] is a tool that distributes a task into a set of computers that make up a rendering farm of computers. It supports some rendering engines as Blender, Maya, Lightwave and so on. The fact of using Dr Queue implies that all resources are centralized and the user has total control over the system.

TeraDRE (TeraGrid Distributed Rendering Environment) [GAST06] describes the implementation of a distributed rendering environment (DRE) by making use of TeraGrid. The main drawback is the complexity to submit works due to the need of dealing with complex scripts and configuration files. In contrast, Yafrid-NG provides an intuitive and easy to use GUI to interact with the system.

There is another approach called BURP [BUR] (Berkeley Ugly Rendering Project), which is a public system that uses distributed systems for rendering scenes. The BURP project is still in testing phase so it is not working at 100%. It is BOINC based and it uses *Volunteer Computing* in order to process the task sent by users. Besides, the whole rendering process is managed by a centralized server which is responsible for assigning work units and gathering the results. So the rendering process is centralized. Furthermore, BURP does not support fine-grained task division and it is not specifically oriented for rendering.

Yafrid-NG addresses the rendering problem from a different point of view, which main characteristics are: i) it is a decentralized architecture. Only a minimal service configuration is necessary. Once a client sends a task, he will manage the whole rendering process, ii) it is completely scalable. Nodes can be easily and dynamically added or removed, iii) it is independent of the hardware and software of nodes that compose the system, iv) a p2p protocol has been proposed for the file exchange. This protocol optimizes this process and avoids the overload for the original file host, v) Resources are managed in a suitable way. It is possible to limit the resources that a client can use. If a client misbehaves, resources are released automatically, vi) It has been completely developed by using GPL technologies so developers can modify, study, or adapt Yafrid-NG to their own needs.

3. Yafrid-NG Architecture

Yafrid-NG defines two basic roles: *providers* and *clients*. *Providers* are computers that hand over computation capability for rendering 3D scenes. A *client* sends works to be processed by the system. Considering that the system must be as decentralized as possible, it is the client himself who manages every necessary issue for the task execution. These issues involve: i) Node coordination for sending the required files. A protocol for file exchange has been defined and the client manages the whole process, ii) Node coordination for

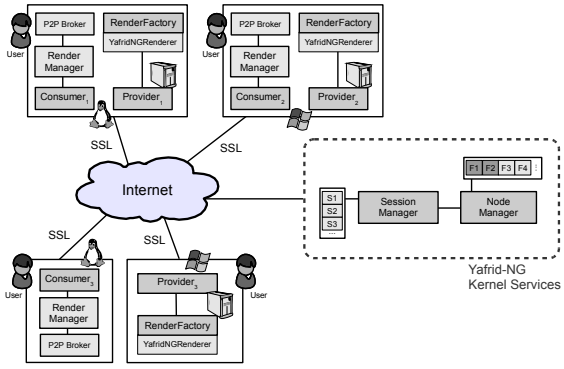

Figura 1: *Basic Yafrid-NG Architecture*

the rendering process. The client who sends the work coordinates the nodes by indicating which work unit has to be performed at each moment.

This is why a client provides two services called *p2pBroker* and *RenderManager* (see Figure 1). The former is used to manage the file exchange and the latter is used to manage the work assignment by avoiding that some work unit be performed more than once, if not necessary.

Another critical issue for the correct management of a grid is the resource management. Clients discover available resources, allocate some resources for a task execution so that these reserved resources are released after the work is done. So, two more services are needed, *SessionManager* and *NodeManager*. To perform a task, a client needs to establish a session with the system via the *SessionManager* service, which keeps track of the established sessions in order to allocate and release resources.

The *NodeManager* keeps track of all the available nodes for rendering and lets adding new resources in a dynamic way. Furthermore, it allows node reservation and their release after the task is completed.

The *SessionManager* and the *NodeManager* are used together to manage and provide access to the system. These are critical services and if any of them fails the whole system is useless. This is why these two services have been replicated by using a Master-Slave approach [New07].

4. Rendering a scene

To render a 3D scene, clients must proceed as follows: i) they must obtain some data about the scene to be rendered, such as resolution, render engine, etc., ii) they must establish a session to the grid, in order to allocate the necessary resources to perform the task, iii) they must transfer the necessary files to the nodes in the system.

Next we describe in detail the architecture and the steps which compose the distributed render process.

4.1. Work Units

A task must be divided into smaller pieces which are then processed by different nodes. We can distinguish two approaches for dividing a task: i) **Fine-grained**, which consists in dividing a frame into smaller pieces, and ii) **Coarse-grained**, which consists in dividing an animation into frames. Whereas the fine-grained approach is suitable to render complex images, the coarse-grained one is suitable to render animations composed of several frames.

The fine-grained approach may produce units of very different complexity. This may be an issue in cases where some work units are much more complex than the average of the others, as the most complex work units condition the render time of the whole task. To avoid this, we propose a partitioning approach that produces work units with similar complexity. To achieve this, we make a fast rasterization of the scene by using the scanline method and an importance function to obtain a grey-scale image (Importance Map). In this, the dark zones represent the less complex areas and the white zones represent the more complex ones.

Due to the randomness of rendering methods, some differences can be perceived after composing the final image. In the static frames, this can be avoided by using an interpolation band. Interpolation band is the common zone shared by two work units. The region comprised between two work units is rendered twice. When the final image is composed, these interpolation bands are overlapped and no difference between work units will be perceived.

4.2. Session Management

Clients use the grid through session objects. A session object is responsible for managing the amount of resources that a client is able to request and keeps track of all engaged nodes, in case they need to be free because of a malfunctioning client. Clients establish a session through the SessionManager and the returned object is interposed between the client and the nodeManager [New06].

The *SessionManager* service is also responsible for the maintenance of the established sessions.

4.3. Node Management

One of the main goals of Yafrid-NG is to dynamically manage the resources that take part into the grid for nodes to be added or removed during the execution time. This makes necessary the *nodeManager* service, which allows to add new resources to the grid. Moreover, this service takes care of reserving resources, so if one of the nodes is being used, it will never be assigned to another client until the node is released. *NodeManager* is an internal service so it is mandatory to establish a session to interact with it.

The *nodeManager* service keeps a list of every available node and the state of the resources (free or busy).

4.4. File Transmission

The client must divide the file into as many parts as nodes participate in the rendering. Since Yafrid-NG follows a p2p approach, nodes communicate directly to each other to get all chunks of data that compose a scene file. Once all nodes have got their corresponding piece, the client will act as a broker for them. When a node gets a new piece, it notifies the client which part of the file has been received. The client keeps track of all nodes and which parts each of them have.

4.5. The rendering process

The *RenderManager* service is the coordinator of the whole rendering process and it provides the mechanisms that control the concurrency and the correct access to shared data. After getting all necessary information, a node starts a loop consisting of i) request for a new zone to be rendered by using the service, *RenderManager*, ii) prepare the parameter for the rendering engine and the desired rendering method, iii) rendering, iv) notify that the zone has already been rendered so the client is able to recover and compose the results. *RenderManager* assigns work units, on demand, to different nodes and when the work unit is done, the *RenderManager* will be notified.

4.6. Getting the results

The process of gathering results consists on taking the file which contains the generated image after a work unit is completed. The process of results retrieval may suppose a considerable amount of time due to the network limitations. To avoid this issue, we use asynchronous calls. One node will use an asynchronous call to notify that some results are available and will continue with its work without waiting for the results to be sent to the client.

When a node indicates that there are new results available, these requests will be stored in a queue to be processed by an independent thread, while the rest of the work is being performed.

To gather results, Yafrid-NG will spawn an independent thread that recovers the desired results from the client. This thread is released when all results have been received or when there are no more operative nodes.

5. Experimental results

We have validated the system described in this paper by a set of experiments that involve rendering static frames and animations. The platform consists of a set of heterogeneous computers connected to a LAN. We employed Intel(R) Core(TM)2 CPU between 1.86GHz and 2.13GHz and LAN bandwidth was 100 Mbps.

In order to prove scalability when rendering animations, we have tested our system with one animation consisting

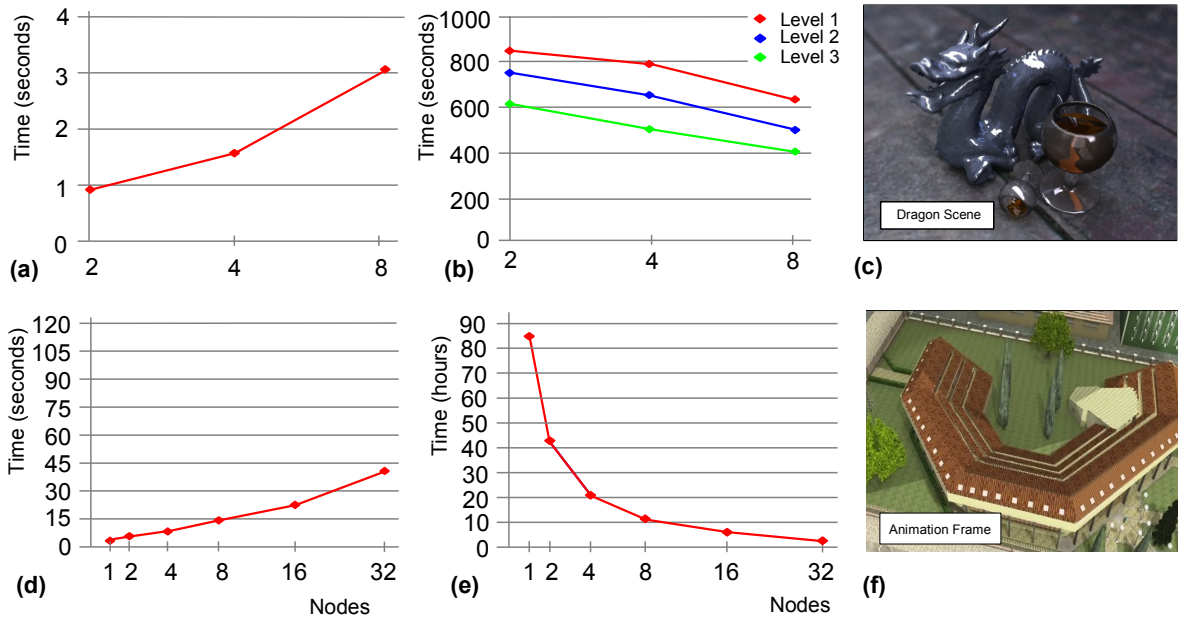

Figura 2: *Experimental results. (a) Transference time for the static frame file; (b) Rendering time for the static frame. (c) Result obtained. (d) Transference time for the animation. (e) Rendering time for the animation. (f) One frame from the generated animation.*

on a virtual visit to an university building. The size of the file is 29.8 MB and it takes 263 MB in RAM when it is uncompressed. The total number of triangles that compose the model is 420,096. The output resolution has been set at 720x576 and the frame rate at 25 frames per second (fps). We have rendered 40 seconds of animation that makes a total of 1000 frames in each test. Figure 2 summarizes the obtained results.

6. Conclusions

This work proposes an infrastructure for the distributed rendering process. It can be composed of heterogeneous and independent machines spread over the Internet.The system is as decentralized as possible. There are just two necessary services that are the *SessionManager* and the *NodeManager*.

7. Acknowledgement

This work has been funded under research projects PII2I09-0052-3440 and PII1C09-0137-6488.

References

[AF06] ANDERSON D. P., FEDAK G.: The computational and storage potential of volunteer computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid* (Washington, DC, USA, 2006), IEEE Computer Society.

[BFH*04] BUCK I., FOLEY T., HORN D., SUGERMAN J., FATAHALIAN K., HOUSTON M., HANRAHAN P.: Brook for gpus: stream computing on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), ACM Press.

[BUR] Burp's official site. http://burp.boinc.dk.

[CDR02] CHALMERS A., DAVIS T., REINHARD E.: *Practical Parallel Rendering*. A. K. Peters, Ltd., 2002.

[DRQ] Dr queue's official site. http://www.drqueue.org.

[FK04] FOSTER I., KESSELMAN C.: *The GRID: Blueprint for a new computing infraestructure*. Morgan Kaufman, 2004.

[GAST06] GOODING S., ARNS L., SMITH P., TILLOTSON J.: Implementation of a distributed rendering environment for the TeraGrid. *IEEE Challenges of Large Applications in Distributed Environments* (2006).

[New06] NEWHOOK M.: Custom sessions and icegrid. *Connections*, 19 (2006).

[New07] NEWHOOK M.: Master-slave replication with ice. *Connections*, 23 (2007).

[SST06] SAMARDZIC A., STARCEVIC D., TUBA M.: An implementation of raytracing algorithm for the multiprocessor machines. *Yugoslav Journal of Operations Research 16*, 1 (2006).