3D Distributed Rendering and Optimization using Free Software

Carlos González-Morcillo[1,2], Gerhard Weiss[2], David Vallejo[1], Luis Jiménez[1]y Jose A. Fdez-Sorribes[1]

[1] Escuela Superior de Informática, University of Castilla-La Mancha
Paseo de la Universidad, 4. Ciudad Real (Spain)
{Carlos.Gonzalez,David.Vallejo,Luis.Jimenez}@uclm.es

[2] Software Competence Center, Hagenberg (Austria)
{Carlos.Morcilla,Gerhard.Weiss}@scch.at

Resumen The media industry is demanding high fidelity images for their 3D synthesis projects. Rendering is the process by means of which a 2D image can be obtained from the abstract definition of a 3D scene. Despite the development of new techniques and algorithms, this process is computationally intensive and requires a lot of time to be done, specially when the source scene is complex or when photo-realistic images are required. This paper describes Yafrid (standing for *Yeah! A Free Render grID*) and MagArRO (*Multi Agent AppRoach to Rendering Optimization*) architectures developed in the University of Castilla-La Mancha for distributed rendering optimization.

Palabras clave: Rendering, Optimization, Artificial Intelligence, Agent.

1. Introduction

Physically based Rendering is the process of generating a 2D image from the abstract description of a 3D scene. The process of constructing a 2D image requires several phases such as modelling, setting materials and textures, placing the virtual light sources and rendering. Rendering algorithms take a definition of geometry, materials, textures, light sources and virtual camera as input and produce an image (or a sequence of images in the case of animations) as output. High-quality photorealistic rendering of complex scenes is one of the key goals of computer graphics. Unfortunately, this process is computationally intensive and requires a lot of time to be done when the rendering technique simulates global illumination. Depending on the rendering method and the scene characteristics, the generation of a single high quality image may take several hours (or even days!).

Because of the huge amount of time required to be done, the rendering phase is often coosidered to be a bottleneck in photorealistic projects in which one image may need some hours of rendering in a modern workstation.

To solve this problem, several approaches based on parallel and distributed processing have been developed. One of the most popular is the render farm: a computer cluster owned by an organization in which each frame of an animation is independently calculated by a single processor. There are new approaches called Computational Grids which uses the Internet to share CPU cycles. Yafrid is a computational Grid that distributes the rendering of a scene among a large number of heterogeneous computers connected to the Internet.

This paper describes the work flow and the free software tools used in the University of Castilla-La Mancha in several 3D rendering projects (based on OSCAR Open Source Cluster Application Resources, and Blender & Yafray render engines), as well as our new research software distributed under the GPL license. Going into detail, the global architecture of Yafrid and the optimization system based on principles from the area of multi-agent system called MagArRO are exposed. This last system uses expert knowledge to make local optimizations in distributed rendering. Finally, some experimental results which show the benefits of using these distributed approaches are presented. The paper is structured as follows. The following section overviews the state of the art and the current main research lines in rendering optimization. Thereby, the focus is on the issues related to parallel and distributed rendering. The next sections describe the general architecture of an Oscar-based cluster, the Grid-based rendering system called Yafrid and the Distributed Intelligent Optimization Architecture called MagArRO. Then, in the next section empirical results that have been obtained by using these systems are shown. The final section is dedicated to a careful discussion and concluding remarks.

1.1. Related Work

There are a lot of rendering methods and algorithms with different characteristics and properties [11,6,10]. However, as pointed out by Kajiya [6], all rendering algorithms aim to model the light behaviour over various types of surfaces, and try to solve the so-called rendering equation which forms the mathematical basis of all rendering algorithms. Common to these algorithms, the different levels of realism are related to the complexity and the computational time required to be done. Chalmers et al. [3] expose various research lines in rendering optimization issues.

Optimizations via Hardware. One alternative to decrease time is making special optimizations using hardware. In this research line there are different approaches; some methods use the programmable GPUs (Graphics Processing Units) as massively parallel, powerful streaming processors which run specialised portions of code of a raytracer. The use of programmable GPUs outperforms the standard workstation CPUs by over a factor of seven [2]. The use of the CPU in conjunction with the GPU requires new paradigms and alternatives to the traditional architectures. For example, the architectural configurations proposed by Rajagopalan et al. [8] demonstrate the use of a GPU to work on real-time

rendering of complex datasets of data which demand complex computations. There are some render engines designed to be used with GPU acceleration, such as Parthenon Renderer [5], which uses the floating-point of the GPU, or the Gelato render engine which works with Nvidia graphic cards.

Optimizations using distributed computing. If we divide the problem into a number of smaller problems (each of them is solved on a separate processor), the time required to solve the full problem would be reduced. In spite of being true in general, there are many distributed rendering problems that would be solved. To obtain a good solution of a problem on a distributed system, all processing elements must be fully utilized. Therefore, a good task scheduling strategy must be chosen. In a domain decomposition strategy [3], each processing unit has the same algorithm, and the problem domain is divided to be solved by the processors. The domain decomposition can be done using a data driven or demand driven approach. In data driven model, the tasks are assigned to the processing units before computation starts. In the other alternative, demand driven model, the tasks are allocated dynamically when the processing units become idle. This is done by implementing a pool of available tasks. This way, the processing units make a request for pending work.

In both models (data and demand driven) is necessary a cost estimation function of each task. This cost prediction is very difficult to be exactly done before the image has been completed due to the nature of global illumination algorithms (unpredictable ray interactions and random path of light samples).

The biggest group of distributed and parallel rendering system is formed by dedicated clusters and rendering farms. Some 3D animation companies use their rendering farms in their own productions and some of them offer rendering services via Internet. According to the way the division of tasks is done, we talk about **fine-grained** systems, in which each image is divided in small parts that are sent to a processor to be done independently, or **coarse-grained** (in case of animations) in which each frame of an animation is entirely done by one processing unit. Dr. Queue [17] is an open source tool designed to distribute frames through a farm of networked computers. This multiplatform software works in coarse-grained division level. In section 2, our solution based on Oscar open cluster [18] is exposed.

New approaches of distributed rendering use a grid design to allocate the tasks among a large number of heterogeneous computers connected to the Internet, using the idle time of the processor [1]. This emerge technology is called *Volunteer Computing* or *Peer-to-peer* computing, and is currently used in some projects based on the BOINC technology (such as BURP [16] (*Big and Ugly Rendering Project*). In section 3, the main architecture of Yafrid and the key advantages are exposed.

Cost prediction. The knowledge about the cost distribution across the scene (i.e. across the different parts of a partitioned scene) can significantly aid the allocation of resources when using a distributed approach. In fact, this estimation

is absolutely necessary in comercial rendering production, to assure deadlines and provide accurate quotations for the work to be done. There are many approaches based on knowledge about cost distribution; a good example is [9]. In section 4.1, the cost prediction mechanism used in MAgArRO is exposed.

Distributed Multi-Agent Optimization. The distribution of multi-agent systems and their properties of intelligent interaction allow us to get an alternative view of rendering optimization. The work presented by Rangel-Kuoppa [7] uses a JADE-based implementation of a multi-agent platform to distribute interactive rendering tasks on a network. Although this work employs the multi-agent metaphor, essentially it does not make use of multi-agent technology itself. In fact, the use of the JADE framework is only for the purpose of realizing communication between nodes, but without knowledge neither learning and negotiation. The MAgArRO architecture proposed in section 4 is an example of a free and Distributed Multi-Agent architecture which employs expert knowledge to optimize the rendering parameters.

2. Oscar-based cluster approach

Nowadays, universities have good practical classrooms provided with plenty of computers. This equipment is maintained and updated frequently. Nevertheless, these computers are inactive over vacation and at night. This existing hardware infrastructure can be coordinated during idle time by using free software and by creating clusters and low-cost supercomputers [14]. Oscar [18] is a software platform which allows the programming of computer clusters based on GNU/Linux. In the next section, the general architecture of the system based on Oscar will be explained. This tool have been used in the University of Castilla-La Mancha to render some 3D projects [20,22].

2.1. Architectural Overview

In our execution environment, the system is composed by 130 workstations placed in different classrooms. These computers form a heterogeneous set of PCs. Every classroom has a specific type of hardware (all based on x86 architecture). The minimal requirements to belong to the system are 500MB of RAM, a swap partition of 1GB, and to have a connection of at least 100Mbits/s (all computers are connected to one network using 100 Mbits/s switches). The figure 1 illustrates these requeriments.

The classrooms, where Oscar cluster is used, are dedicated to educational activity. For this reason, it was most appropiate not to install permanently any software in them. The subproject Thin-Oscar [19] allows us to use machines without local HD or a partition to install the operating system as members of the Oscar cluster. A Swap partition is used for the management of temporal files (used in the rendering of each project).

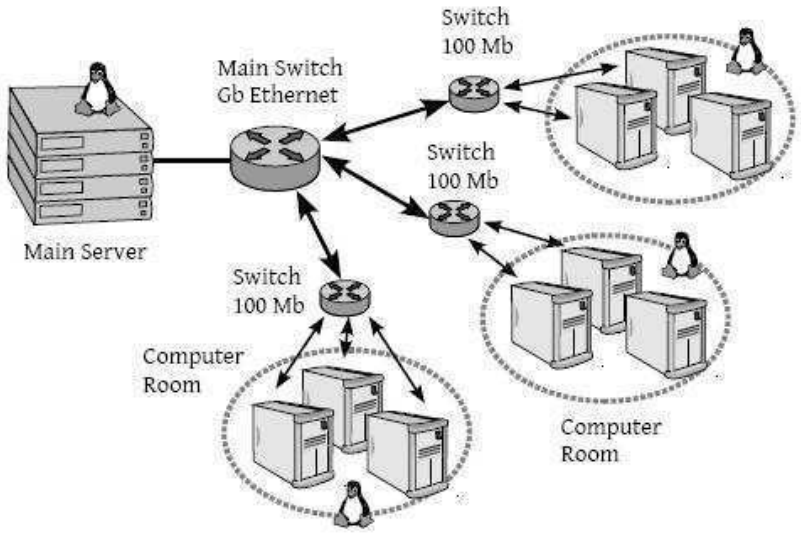

Figura 1. Oscar-based rendering farm at ESI-UCLM.

Each rendering node is configured by obtaining the configuration parameters from the network. This is done by using the PXE extension of the BIOS (Pre eXecution Environment). In our case, these data are the Operating System image in which will be executed. This way, without installing anything in the hard disk, the minimum image of the operating system is loaded (only the basic services and the render engine).

The server has two key processes to handle the PXE requests:

- **DHCPD:** It is the Dinamic Host Configuration Protocol daemon. This protocol is used to tell the client its IP and the operating system image to load.
- **TFTPD:** It is the Trivial Transfer Protocol daemon. When the server receives a file request, sends it to the client by using some configuration tables.

To begin and finish the execution of the computers in a controlled schedule, the WOL (Wake On Lan) functionality of modern computers is used. These BIOS extensions are used with the help of the motherboard and the software package Ether-Wake (developed by Donal Becker). Even if the computer is off, the network card continues listening. When the package generated by Ether-Wake arrives, the computer boots and by means of a PXE request, it loads the Operating System image. All the machines are in the same broadcast domain of the network. To avoid possible intrusions, all the machines which are being used in Oscar belong to a specific VLAN separated from the rest of the organization.

Finally, to halt the computers when the allowed time has finished, the ACPI interface must be correctly configured. To use this functionality, a simple script in the server side was made. The server establish a ssh connection to each node and send it a *shutdown* command.

3. Yafrid: Grid-based Rendering

Yafrid is basically a system which takes advantage of the characteristics of the computational grids by distributing the rendering of a scene throught the Internet. The system also has other important tasks related to the management of the workunits and the controlled use of the grid.

3.1. Architectural Overview

The top-level components of Yafrid are basically the following ones:

- **Server.** The hub of Yafrid. Its basic component is the Distributor which takes works from a queue and sends them to the providers.
- **Service Provider.** This entity process the requests of the clients.
- **Client.** A client is an external entity which doesn't belongs to the system in a strict sense. Its role in the operation of the grids consists in submitting works to be done by the providers. Those works are stored in a queue from where distributor will take the next one to be scheduled.

In terms of access to the system, three user roles have been defined which determine the access privileges that the user has. Those user roles are:

- **Client.** Having this role allows an user to submit works to the grid. A client is also able to create and manage render groups (clients and providers can subscribe to these groups). When a project is created, it can belong to a group. In this case, only providers belonging to the same group can take part in the rendering of the project.
- **Administrator.** This user is necessary to operate the whole system and has complete privileges to access to the information about all the users of the system.
- **Provider.** The provider is a user that has installed the necessary software to allow the grid to sent works. Providers can access their own information and some statistics.

Yafrid server The server is the fundamental node around of which the whole Yafrid render system is established. Each one of the providers connects to this node in order to let the grid use its CPU cycles for the rendering of the scenes submitted by Yafrid clients.

Yafrid server is developed over an architecture in which, in general, four layers can be distinguished (Figure 2). This design is slightly based on the architecture

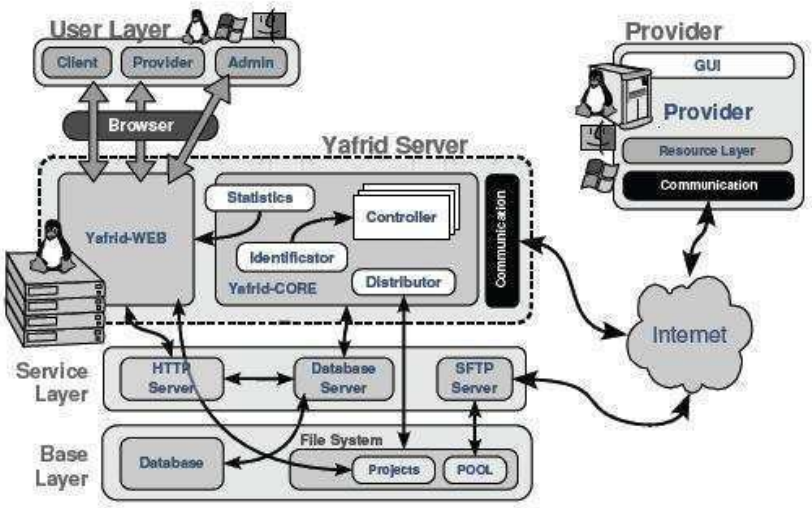

Figura 2. Yafrid General Architecture.

that appears in [4]. Those layers are *'Resource Layer'*, *'Service Layer'*, *'Yafrid Server'* and *Úser Layer'* from lowest to highest level of abstraction.

Resource Layer. This layer has the lowest abstraction level and it is the most related with the operating system issues. Resource layer has the following components:

- **Database system.** It is in this database where the necessary tables for the correct operation of the system are maintained. Some of these tables are used to obtain statistics about the system performance while other ones store the data associated to users, groups, projects, etc. This database is accessed from the levels above by means of a database server. The current implementation uses MySQL.
- **Filesystem.** Sometimes, it is necessary to access directly to the filesystem from the layers above. Basically, the system distinguishes two types of directories. There are some directories which are used to store the workunits of the launched projects that will be accessed via SFTP by providers. Those directories compose the workunits POOL. The other category of directories is composed by those ones that contains the information about the users and their projects.
- **Network system.** The module dedicated to the communications that belongs to the main layer hides the utilization of the network resources of the computer by using a middleware (the current implementation uses ICE).

Service Layer. Basically, this layer contains the different servers that allow modules from the layers above to access the resources that belongs to the layer under this one. There are the following servers in this level:

- **HTTP Server.** Yafrid-WEB module is established over this server. As Yafrid-WEB has been developed using dynamic web pages written in a web-oriented scripting language (the current implementation has been done using PHP), the web server has to have support for this language. It is also necessary to have support for composition of graphics and for accessing to the database.
- **Database server.** This server is used by the different modules of Yafrid to access the indispensable data for the system operation.
- **SFTP server.** Accessed by the service providers to obtain the necessary files to carry out the rendering of the workunits. Once the rendering has finished, the SFTP server will be used to send to the Yafrid server the resultant image.

Yafrid Layer This is the main layer of the server and it is composed by two different modules (Yafrid-WEB and Yafrid-CORE) which work independently one of the other. **Yafrid-WEB** is the interactive module of the server and it has been developed as a set of dynamic web pages written using HTML and a wed-oriented scripting language. **Yafrid-CORE** is the non-interactive part of the server. This module has been mainly developed using Python and . Yafrid-CORE is composed by three submodules; Distributor, Identificator and Statistics.

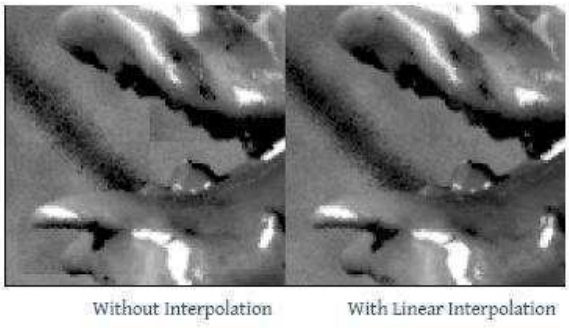

Figura 3. Artifacts without interpolation between workunits.

The **distributor** is the active part of the server. It implements the main algorithm that is aimed to do the following indispensable tasks such as generating the workunits, assigning the generated workunits to providers, controlling the Timeout, finishing projects and composing the results. With the results generated by the different providers, the distributor has to compose the final image.

This process is not trivial because slight differences between fragments can be distinguished when obtained from distinct computers due to the random component of Monte Carlo based methods (like Pathtracing). For that reason, it is necessary to smooth the joint between fragments which are neighbors using a lineal interpolation mask. We define a zone in the workunit that is used to combine with other workunits in the server. In Figure 3 on the left, we can see problems when joining the workunits if we don't use a blending method.

The passive part of Yafrid-CORE is called the **Identificator** module whose mission consists in waiting for the communications from the providers. The first time a provider try to connect to the Yafrid server the Identificator generates an object, the provider controller, and returns a proxy to this object. Each provider has its own controller.

Provider. The provider is the software that the users who wants to give CPU cycles to be used by the grid in rendering tasks must to be installed in the computer. It can work in both visual and non-visual mode. The first thing that a provider has to do is to connect to the grid. Once activated, the provider waits until the server sends a workunit to be processed. After finishing the rendering, the provider sends the file via SFTP and tell the controller the work is done.

4. MAgArRO: Distributed Intelligent Optimization

According to [12], an agent is a computer system that is situated in some environment and that is capable of autonomous action ini this environment in order to meet its design objectives. MAgArRO uses the principles, techniques and concepts known from the area of multi-agent systems, and is based on design principles of the FIPA [21] standards.

MAgArRO is also developed using the ICE middleware [25]. The location service IceGrid is used to indicate in which computer the services reside. Glacier2 is used to solve the difficulties related with hostile network environments, making available agents connected through a rotuer and a firewall.

4.1. Architectural Overview

As mentioned, the overall architecture of MAgArRO is based on the design principles of the FIPA standard. In figure 4 the general workflow and main architectural roles are shown. in addition to the basic FIPA services, MAgArRO includes specific services related to Rendering Optimization. Specifically, a service called Analyst studies the scene in order to enable the division of the rendering task. A blackboard is used to represent some aspects of the common environment of the agents. Finally, a master service called *Master* handles dynamic groups of agents who cooperate by fulfilling subtasks.

Figure 4 also illustrates the basic workflow in MAgArRO (the circled numbers in this figure represent the following steps). **1** – The first step is the subscription of the agents to the system. This subscription can be done at any moment;

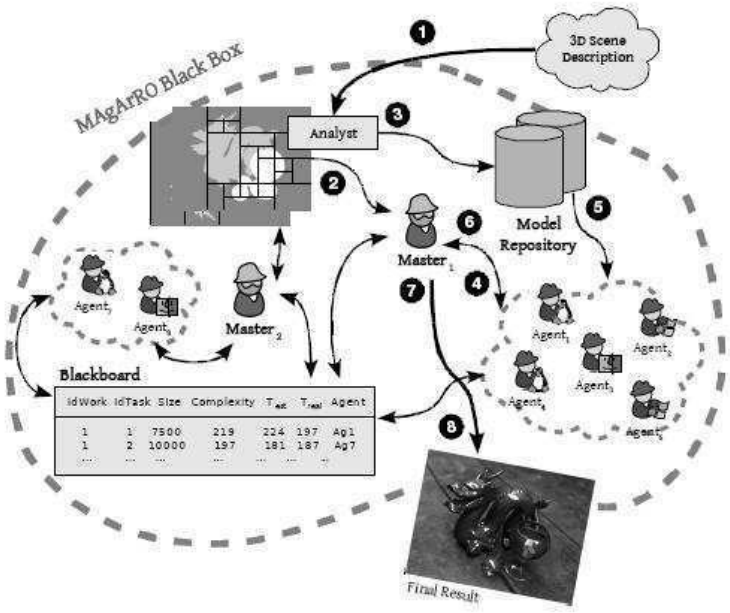

Figura 4. General workflow and main architectural roles.

the available agents are managed dynamically. When the system receives a new file to be rendered, it is delivered to the Analyst service. **2** – The Analyst analyzes the scene, making some partitions of the work and extracting a set of tasks. **3** – The Master is notified about the new scene which is sent to the Model Repository. **4** – Some of the agents available at this moment are managed by the Master and notified about the new scene. **5** – Each agent obtains the 3D model from the repository and an auction is started. **6** – The (sub-)tasks are executed by the agents and the results are sent to the Master. **7** – The final result is composed by the Master using the output of the tasks previously done. **8** – The Master sends the rendered image to the user. Key issues of this workflow are described in the following.

Analysis of the Scene using Importance Maps. MAgArRO employs the idea to estimate the complexity of the different tasks in order to achieve load-balanced partitining. Complexity analysis is done by the Analyst agent prior to (and independent of) all other rendering steps.

The main objective in this partitioning process is to obtain tasks with similar complexity to avoid the delay in the final time caused by too complex tasks. This analysis may be done in a fast way independently of the final render process.

Once the importance map is generated, a partition is constructed to obtain a final set of tasks. These partitions are formed hierarchically at different levels,

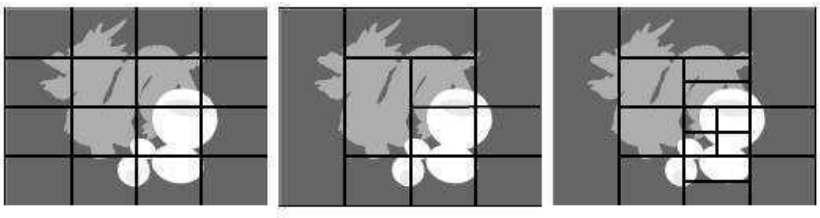

Figura 5. Importance maps.*Left:* Blind partitioning (First Level). *Center:* Join zones with similar complexity (Second Level). *Right:* Balancing complexity/size ratio (Third Level).

where at each level the partitioning results obtained at the previous level are used. At the first level, the partition is made taking care of the minimum size and the maximum complexity of each zone. With these two parameters, the *Analyst* makes a recursive division of the zones (see Figure 5). At the second level, neighbor zones with similar complexity are joined. Finally, at the third level the *Analyst* tries to obtain a balanced division where each zone has nearly the same complexity/size ratio. The idea behind this division is to obtain tasks that all require roughly the same rendering time. As shown below in the experimental results, the quality of this partitioning is highly correlated to the final rendering time.

Using Expert Knowledge. When a task is assigned to an agent, a fuzzy rule set is used in order to model the expert knowledge and optimize the rendering parameters for this task. Fuzzy rule sets are known to be well suited for expert knowledge modeling due to their descriptive power and easy extensibility [13]. The output parameters (i.e. the consequent part of the rules) are configured so that the time required to complete the rendering is reduced and the loss of quality is minimized. Each agent may model different expert knowledge with a different set of fuzzy rules. For example, the following rule is used (in a set of 28 rules) for describing the rendering parameters of Pathtracing method:

R_1**: If** C is {B,VB} $\wedge$ S is B,N $\wedge$ Op is VB **then** Ls is VS $\wedge$ Rl is VS

The meaning of this rule is "If the Complexity is Big or Very Big and the Size is Big or Normal and Optimization Level is Very Big then the number of Light Samples is Very Small and the Recursion Level is Very Small". The Complexity parameter reepresents the complexity/size ratio of the task, the Size represents the size of the task in pixels, the Optimization Level is selected by the user. The output parameter Recursion Level defines the global recursion level in raytracing (number of light bounces) and the Light Samples defines the number of samples per light in the scene (the biggest, the more quality and the higher rendering time).

5. Experimental Results

In order to test the behaviour of the systems, 8 computers with the same characteristics were connected to Yafrid and MAgArRO. These nodes (Intel Pentium Centrino 2 GHz, 1GB RAM) were used in both systems during the execution of all tests. The test scene contained more than 100.000 faces, 5 levels of raytracing recursion in mirror surfaces (the dragon), 6 levels in transparent surfaces (the glass), and 128 samples per light source was rendered using the free render engine Yafray [23]. In addition, 200.000 photons were shooted in order to construct the Photon Map structure. With this configuration, the rendering on a single machine without any optimization took 121:17 (121 minutes and 17 seconds).

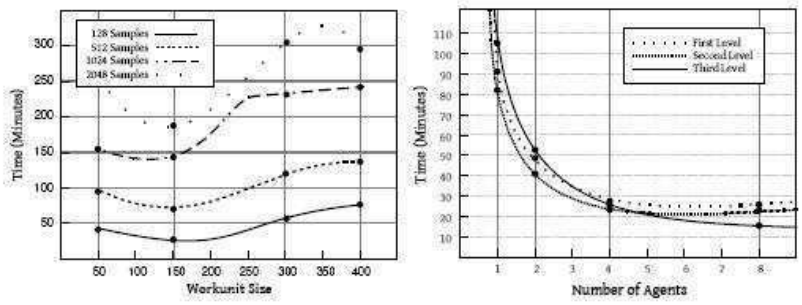

Figura 6. *Left:* Yafrid. Rendering time related to Workunit size. *Right:* MAgArRO. Different levels of partitioning with Normal optimization level.

In the case of Yafrid, as we can see in Figure 6 *(Left)*, the rendering time in the best case is near seven times better using the grid and less than twice in the worst case. With these results it is clear the importance of choosing an appropiate workunit size. This occurs because there are complex tasks that slow down the whole rendering process even if the number of nodes is increased.

As we mentioned, MAgArRO uses *Importance Maps* to estimate the complexity of the different tasks. Figure 6 *(Right)* shows the time required by using different partitioning levels. Using a simple first-level partitioning (similar to the Yafrid approach), a good render time can be obtained with just a few agents. However, when the number of agents (processing nodes) grow, the overall performance of the system increases because the differences in the complexity of the tasks are relatively small.

As a final remark, note that intelligent optimization may result in different quality levels for different areas of the overall scene. This is because more aggresive optimization levels (Big or Very Big) may result in a loss of detail.

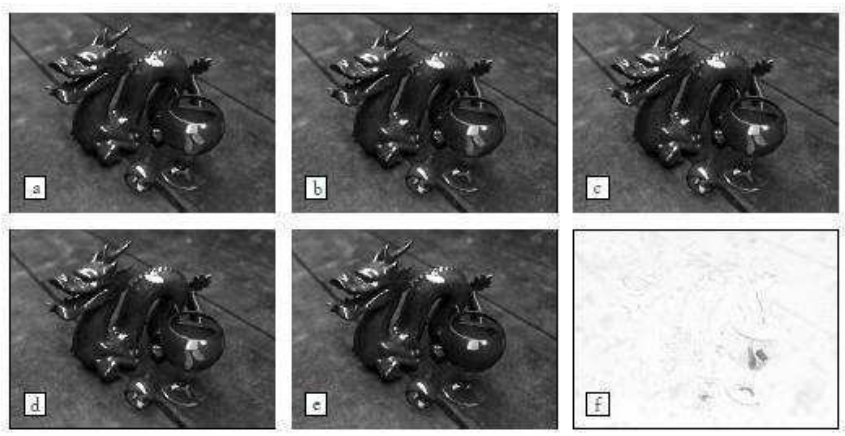

Figura 7. Result of the rendering using different optimization levels. (a) No optimization and render in one machine. (b) Very Small (c) Small (d) Normal (e) Very Big (f) Difference between (a) and (e) (the lighter colour, the smaller difference).

For example, in Figure 7.e, the reflections on the glass are not so detailed as in Figure 7.a. The difference between the optimal render and the most aggresive optimization level (Figure 7.f) is minimal.

6. Discussion and Conclusion

The computational requeriments of photo-realistic rendering are huge and, therefore, to obtain the results in a reasonable time and on a single computer is practically impossible (even more difficult in the case of animations). Several approaches based on different technologies have been exposed in this paper.

Our cluster based on the **Oscar** system has some interesting characteristics:

- Very good throughput in the case of animations. The system divides each frame of the animation in different nodes of the cluster. The fine-grained approach needs the programming of new features in the main server.
- The processing nodes are used during the idle time (at night).
- The latency due to the file transfer is minimal (thanks to the use of a Fast Ethernet network).

Otherwise, the cluster can only be used submitting tasks to the main server into the same organization.

To solve some of these problems, the **Yafrid** approach was designed. This computational grid has some important advantages:

- There is no cluster; the providers can be heterogeneous (software and hardware) and can be geographically distributed.

- With the fine-grained approach, we can make local optimizations in each frame.
- One of the main advantages of this distributed approach is the scalability. The performance perceived by the user depends on the number of subscribed providers.

Some enhancements would be done to improve the performance of Yafrid. Some of them was added to **MAgArRO**:

- MAgArRO enables importance-driven rendering throught the use of importance maps.
- It allows the application of expert knowledge by employing flexible fuzzy rules.
- It applies the principles of decentralized control and local optimization. The services are easily replicable, thus possible bottlenecks in the final deploy can be minimized.

There are many future research lines. In our current work, we concentrate on the combination of the best characteristics of Yafrid and MAgArRO to integrate the new system (called YafridNG) in the official Blender branch [15]. The source code of these systems, distributed under GPL license, can be downloaded from [24].

7. Acknowledgments

This work has been funded by the *Consejeria de Ciencia y Tecnología* and the *Junta de Comunidades de Castilla-La Mancha* under Research Projects PAC-06-0141 and PBC06-0064. Special thanks to Javier Ayllon for his support from the Supercomputation Service (University of Castilla-La Mancha).

Referencias

1. Anderson, D.P., Fedak, G.: The Computational and Storage Potential of Volunteer Computing. Sixth IEEE International Symposium on Cluster Computer and the Grid (CCGRID '06). 73–80. May 2006.
2. Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., Hanrahan, P.: Brook for GPUs: Stream Computing on Graphics Hardware. Proceedings of SIGGRAPH '04, 777–786.
3. Chalmers, A., Davis, T., Reinhard, E.: Practical Parallel Rendering. Ed. A. K. Peters, 2002. ISBN: 1-56881-179-9.
4. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputing Applications 15, 3(2002).
5. Hachisuka, T.: High-Quality Global Illumination Rendering using Rasterization. GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation. Addison-Wesley Professional, 2005.

6. Kajiya, J. T.: The rendering equation. Computer Graphics 20(4): 143–150. Proceedings of SIGGRAPH '86.
7. Kuoppa, R. R., Cruz, C. A., Mould, D.: Distributed 3D Rendering System in a Multi-Agent Platform. Proceedings of the Fourth Mexican International Conference on Computer Science, 8, 2003.
8. Rajagopalan, R., Goswami, D., Mudur, S. P.: Functionality Distribution for Parallel Rendering. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), 18–28, April 2005.
9. Reinhard, E., Kok, A. J., Jansen, F. W.: Cost Prediction in Ray Tracing. Rendering Techniques '96, 41–50. Springer-Verlag, June 1996.
10. Veach, E., Guibas, L. J.: Metropolis light transport. Proceedings of SIGGRAPH '97, 65–76. New York, USA: ACM Press - Addison Wesley Publishing Co.
11. Whitted, T.: An improved illumination model for shaded display. Proceedings of SIGGRAPH '79, 14. New York, USA: ACM Press.
12. Wooldridge, M. J.: An introduction to multiagent systems. John Wiley & Sons, 2002. ISBN: 0-471-49691-X
13. Zadeh, L. A.: The concept of a linguistic variable and its applications to approximate reasoning. Information Science, 1975.
14. Beowulf: Open Scalable Performance Clusters. http://www.beowulf.org
15. Blender: Free 3D content creation suite. http://www.blender.org
16. BURP: Big Ugly Rendering Project. http://burp.boinc.dk/
17. Dr. Queue.: OS Software for Distributed Rendering. http://www.drqueue.org/
18. OSCAR: Open Cluster Group. http://www.openclustergroup.org/
19. Thin-Oscar: http://thin-osccar.sourceforge.net/
20. Virtual Tour ESI UCLM. http://www.inf-cr.uclm.es/virtual/index_en.html
21. FIPA. Foundation for Intelligent Physical Agents. http://www.fipa.org
22. Virtual Visit - Hospital Ciudad Real. http://dev.oreto.inf-cr.uclm.es/www/vvhosp
23. Yafray: Yet Another Free Raytracer http://www.yafray.org
24. Yafrid Next Generation. http://www.yafridng.org
25. ZeroC ICE Middleware http://www.zeroc.com