3D Distributed Rendering and Optimization using Free Software

Carlos González-Morcillo, Gerhard Weiss, David Vallejo-Fernández, and Luis Jiménez-Linares, and Javier Albusac-Jiménez

The media industry is demanding high fidelity images for 3D synthesis projects. One of the main phases is Rendering, the process in which a 2D image can be obtained from the abstract definition of a 3D scene. Despite developing new techniques and algorithms, this process is computationally intensive and requires a lot of time to be done, especially 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, which have been developed at the University of Castilla-La Mancha for distributed rendering optimization.

Keywords: Artificial Intelligence, Intelligent Agents, Optimization, Rendering.

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 including 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!). For this reason, the rendering phase is often considered as a bottleneck in photorealistic projects.

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 use the Internet to share CPU cycles. In this context, 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 at the University of Castilla-La Mancha in several 3D rendering projects based on Open Source Cluster Ap-

Authors

Carlos Gonzalez-Morcillo is an assistant professor and a Ph.D. student in the ORETO research group at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, distributed rendering, and fuzzy logic. He received both B.Sc. and M.Sc. degrees in Computer Science from the University of Castilla-La Mancha in 2002 and 2004 respectively. <carlos.gonzalez@uclm.es>.

Gerhard Weiss is the scientific director at SCCH (Software Competence Center Hagenberg GmbH), one of Austria's largest independent research centres. His main interests have been in computational intelligence and autonomous systems in general, and in the foundations and application of agent and multi-agent technology in particular. He is the co/editor of the reference book in this area "Multiagent Systems" (MIT Press). <gerhard.weiss@scch.at>.

David Vallejo-Fernandez is an assistant professor and a Ph.D. student in the ORETO research group at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, cognitive surveillance architectures, and distributed rendering. He received his B.Sc. degree in Computer Science from the University of Castilla-La Mancha in 2006. <david.vallejo@uclm.es>

Luis Jimenez-Linares is an Associate Professor of Computer Science at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, knowledge representation, ontology design, and fuzzy logic. He received both M.Sc. and Ph.D. degrees in Computer Science from the University of Granada in 1991 and 1997 respectively. He is member of the European Society of Fuzzy Logic and Technology. <luis.jimenez@uclm.es>.

Javier Albusac-Jimenez is a researcher and a Ph.D. student in the ORETO research group at the University of Castilla-La Mancha. His recent research topics are multi-agent systems, cognitive surveillance, and cognitive vision. He received his B.Sc. degree in Computer Science from the University of Castilla-La Mancha in 2005. <javieralonso.albusac@uclm.es>.

plication Resources (OSCAR) and Blender & Yafray render engines), as well as our new research software distributed under General Public Licence (GPL). Going into detail, the global architecture of Yafrid and the optimization system (based on principles from the area of multi-agent systems) 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. 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 many rendering methods and algorithms, each having 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 method to decrease time is to make special optimizations using hardware. In this research line there are different approaches; some methods use programmable GPUs (Graphics Processing Units) as massively parallel, powerful streaming processors which run specialised code portions of a raytracer. The use of programmable GPUs out-performs 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 data sets which demand complex computations. There are some render engines designed to be used with GPU acceleration, such as Parthenon Renderer [5], which use 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 being 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 to a full 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 a demand driven approach. In a data driven model, the tasks are assigned to the processing units before starting to compute. In the other alternative, the demand driven model, the tasks are dynamically allocated 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), a cost estimation function of each task is needed. This cost prediction is very difficult to exactly calculate before completing the image due to the nature of global illumination algorithms (unpredictable ray interactions and random paths of light samples).

The biggest group of distributed and parallel rendering systems is formed by dedicated clusters and rendering farms used by some 3D animation companies. Depending on the task division, we can talk about fine-grained systems, in which each image is divided into small parts that are sent to a processor to be independently done, or coarse-grained (in case of animations) in which each frame of an animation is entirely done by one processing unit. In this context, Dr. Queue [17] is an open source tool designed for distributing frames through a farm of networked computers. This multiplatform software works in a 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 emerging 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 its 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. This estimation is absolutely necessary in commercial rendering productions, to assure deadlines and provide accurate quotations. 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, it does not make use of multi-agent technology itself. The MAgArRO architecture proposed in Section 4 is an ex-

ample 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 frequently maintained and updated. Nevertheless, these computers are inactive over vacation and at night. This existing hardware infrastructure can be co-ordinated during idle time by using free software thus creating clusters and low-cost supercomputers [14]. OSCAR [18] is a software platform which allows the user to deploy clusters based on GNU/Linux. In the next section, the general architecture of the OSCAR-based system will be explained. This tool is being used at the University of Castilla-La Mancha to render 3D projects [20][22].

2.1 Architectural Overview

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

The classrooms, where OSCAR cluster is used, are dedicated to education. For this reason, the best choice is not to permanently install any software in them. The subproject Thin-OSCAR [19] allows us to use machines without a local HD or a partition to install the operating system as members of the OSCAR cluster.

Each rendering node is configured obtaining the configuration parameters from the network. This is done by using the Pre eXecution Environment (PXE) extension of the BIOS. In our case, these data are the operating system image in which will be executed.

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

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

In order 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). When the package generated by Ether-Wake arrives, the computer boots and loads the operating system image.

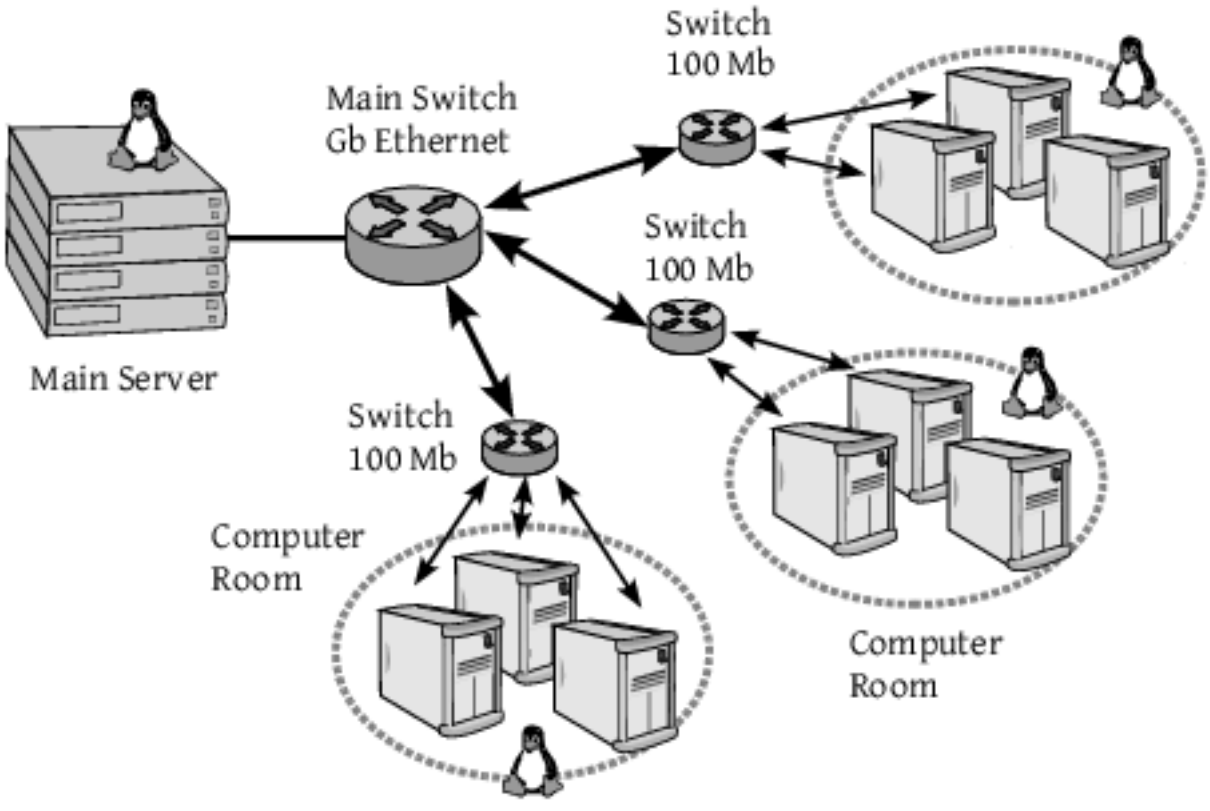

Figure 1: OSCAR-based Rendering Farm at ESI-UCLM.

has finished, the ACPI interface is used to halt them. The server establishes a ssh connection to each node and sends it a *shutdown* command.

3 Yafrid: Grid-based Rendering

Yafrid is basically a system which takes advantage of the characteristics of computational grids by distributing the rendering of a scene through 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 gets works from a queue and sends them to the providers.
- **Service Provider.** This entity processes the client requests.
- **Client.** A client is an external entity which does not belong to the system in a strict sense. Its role is to submit works to the providers. Those works are stored in a queue used by the distributor to take the next one to be scheduled.

In terms of access to the system, three user roles have been defined to determine the user access privileges:

- **Client.** With this role, a user is allowed 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 project rendering.
- **Administrator.** This role is needed for operating the whole system and has complete privileges to access to the information about all the users.
- **Provider.** The provider is a role user that has installed the software needed for receiving works. Providers can access to their own information and some statistics.

Yafrid server. The server is the fundamental node for setting the Yafrid render system up. Each one of the providers connects to this node in order to let the grid to use its CPU cycles for rendering the scenes submitted by Yafrid clients. Yafrid server consists of an architecture of four layers (Figure 2). This design is loosely based on the architecture that appears in [4]. Those layers are "*Resource Layer*", "*Service Layer*", "*Yafrid Server*", and "*User Layer*" (from lowest to highest level of abstraction).

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

- **Database system.** It is in this database where the tables needed for the correct operation of the system are maintained. Some of these tables are used to obtain statistics about the system performance, whereas other ones store the data associated to users, groups, projects, etc. The current implementation uses MySQL.

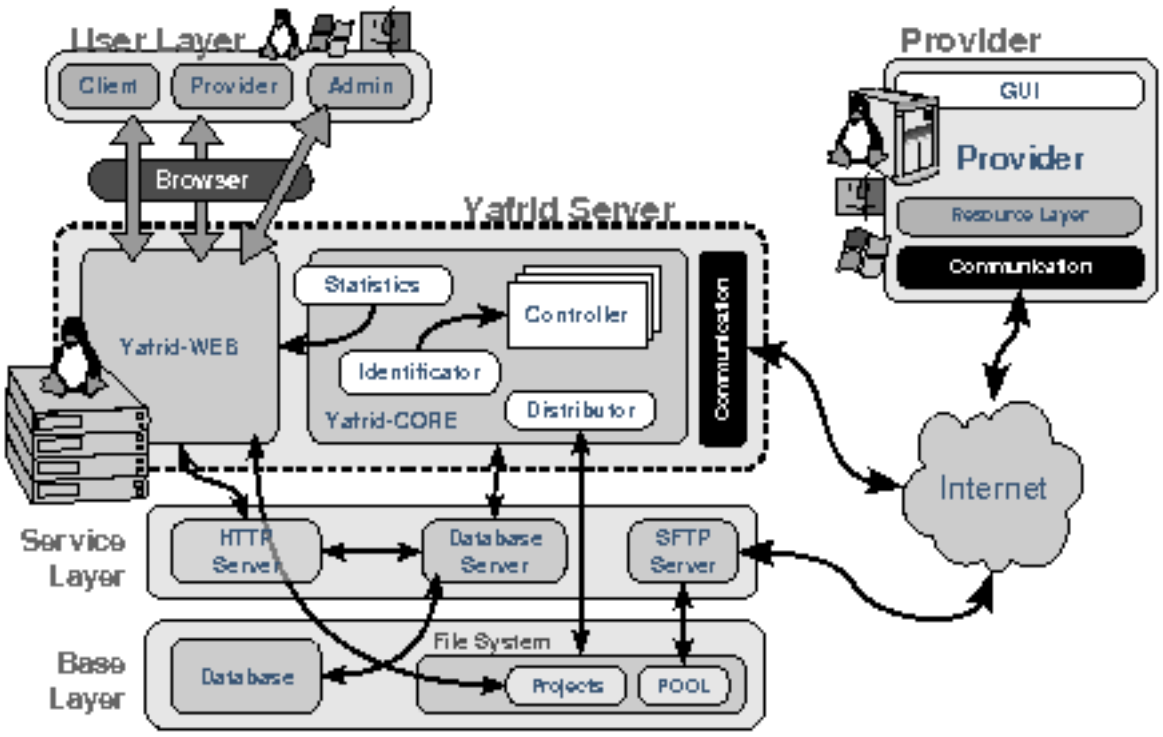

Figure 2: Yafrid General Architecture.

- **Filesystem.** Sometimes, it is necessary to directly access the file system from the high-level layers. Basically, the system distinguishes two types of directories. There are some directories which are used to store the workunits of projects that will be accessed via SFTP by providers. Those directories compose the workunit POOL. The other category of directories is composed by those directories that contain information about users and projects.
- **Network system.** The module dedicated to communications hides the use of network resources by using a middleware (the current implementation uses ZeroC ICE [25]).

Service Layer. Basically, this layer contains the different servers that allow modules to access resources that belong to lower layers. There are several servers at this level:

- **HTTP Server.** The 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 uses PHP), the web server must support this language.
- **Database server.** This server is used by the different Yafrid modules to access to the indispensable data for the system operation.
- **SFTP server.** This server is accessed by the service providers to obtain the files needed for carrying out the rendering of the work units. Once the rendering has finished, the SFTP server will be used to send the resultant image to the Yafrid Server.

Yafrid Layer. This is the main layer of the server and it is composed of two different modules (Yafrid-WEB and Yafrid-CORE) working independently. **Yafrid-WEB** is the interactive module of the server and it has been developed as a set of dynamic web pages. **Yafrid-CORE** is the non-interactive part of the server. This module has been mainly developed using Python. Yafrid-CORE is composed of three submodules: Distributor, Identificator, and Statistics.

- The **Distributor** is the active part of the server. It implements the main algorithm in charge of doing the indispensable tasks, such as generating the work units, assigning them to providers, controlling the timeout, finishing projects, and composing the results. With the results generated by the different providers, the distributor composes the final image. This process is not trivial because slight differences between fragments obtained from different computers can be distinguished (due to the random component of Monte Carlo based methods as Path-tracing). For that reason, it is necessary to smooth the joint between fragments which are neighbours using a lineal interpolation mask. We define a zone in the work unit that is combined with other work units in the server. In Figure 3 on the left, we can see problems when joining the work units if we do not use a blending method.
- The passive part of Yafrid-CORE is called the **Identificator** module. Its mission consists of waiting for the communications from the providers. The first time a provider tries 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 used by the users who want to give CPU cycles to the grid. It can work in both visual and non-visual mode. First, the provider must connect to the grid. Once activated, the provider waits until the server sends a work unit to process. After finishing the rendering,

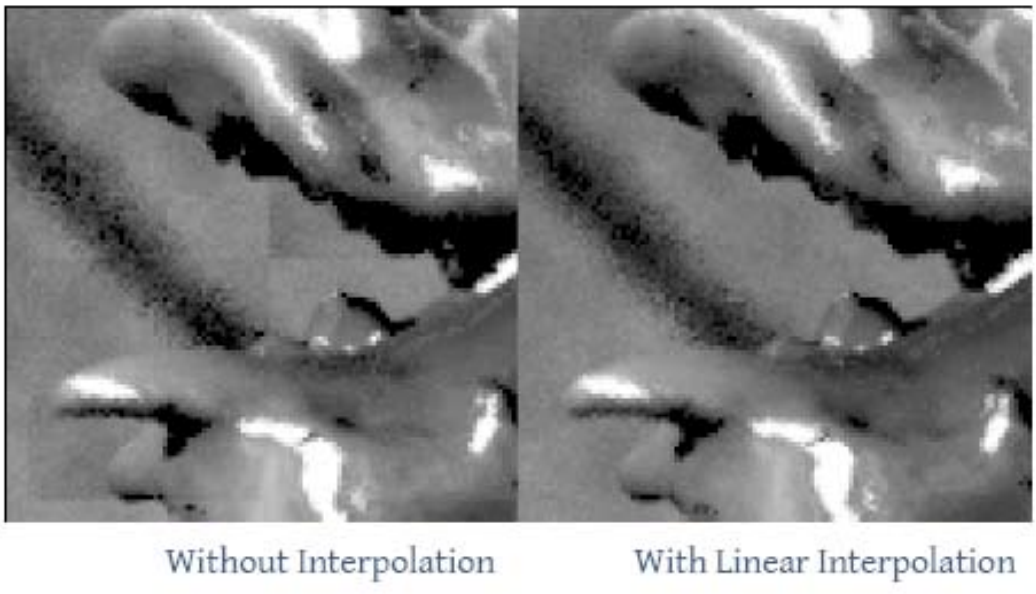

Figure 3: Artifacts without Interpolation between Workunits.

the provider sends the file via SFTP and informs the controller that the work was 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 actioning in 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 it is based on the design principles of FIPA (Foundation for Intelligent Physical Agents) standards [21].

MAgArRO has also been 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, being the agents able to connect behind a router or a firewall.

4.1 Architectural Overview

As mentioned, the overall architecture of MAgArRO is based on the design principles of FIPA standards. In Figure 4, the general workflow and the 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 tasks. 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; the available agents are dynamically managed. When the system receives a new file to be rendered, it is delivered to the Analyst service.

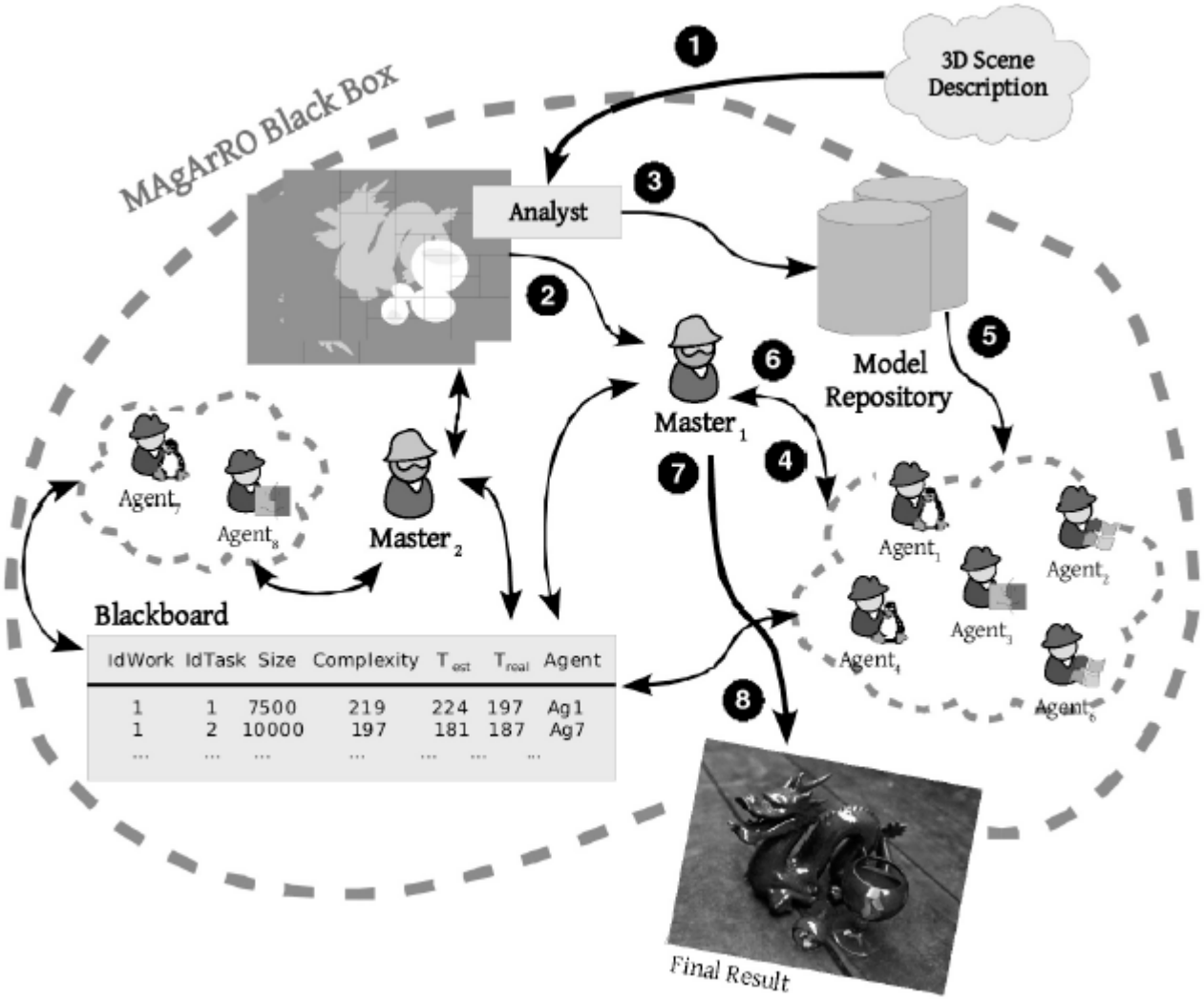

Figure 4: General Workflow and Main Architectural Roles.

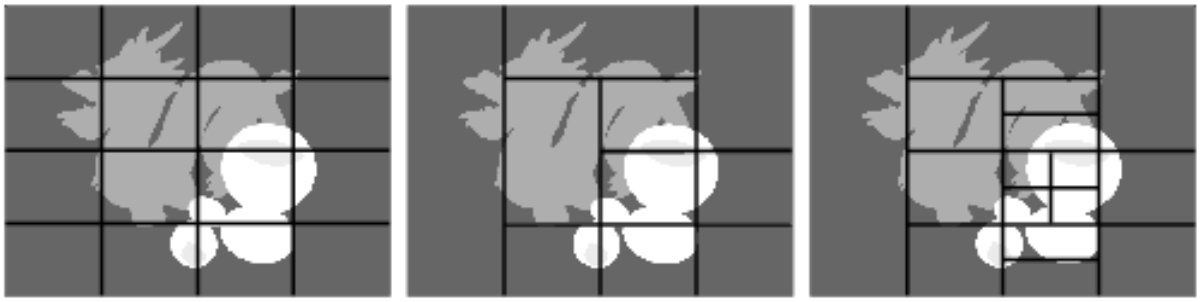

Figure 5: Importance Maps. Left: Blind Partitioning (First Level). Center: Join Zones with Similar Complexity (Second Level). Right: Balancing Complexity/Size Ratio (Third Level).

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 begins to auction.

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 section.

Analysis of the Scene using Importance Maps. MAgArRO employs the idea of estimating the complexity of the different tasks in order to achieve load-balanced partitioning. 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 hierarchically formed at different levels, 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, neighbour 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 set of fuzzy rules is used to model the expert knowledge and to optimize the rendering parameters for this task. Sets of fuzzy rule are considered well suited for expert knowledge modelling 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 the Pathtracing method: R_1: **If** C is {B,VB} and S is {B,N} and Op is VB **then** Ls is VS and Rl is VS.

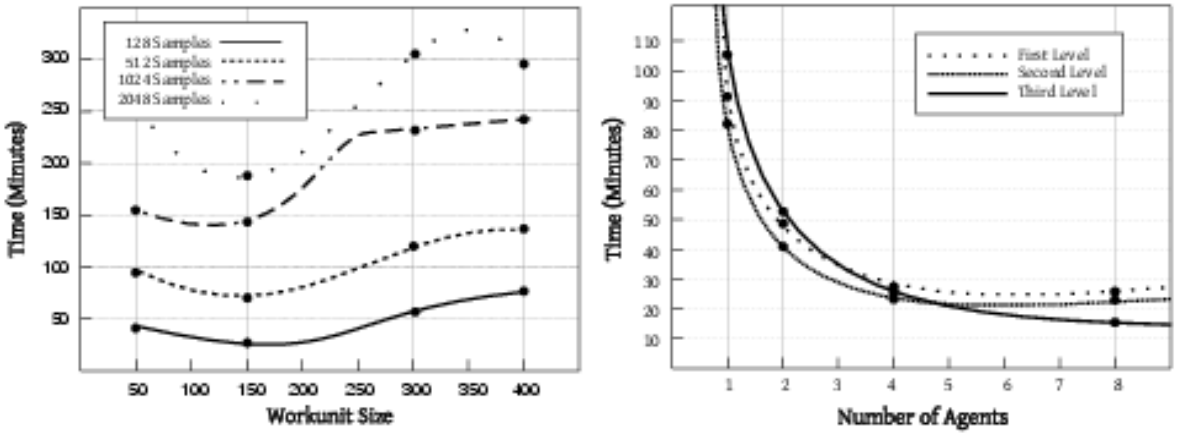

Figure 6: Left: Yafrid. Rendering Time Related to Workunit Size. Right: MAgArRO. Different Levels of Partitioning with a Normal Optimization Level.

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 represents the complexity/size ratio of the task, the Size represents the size of the task in pixels, and 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 (higher values involve more quality and more 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 the 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), 128 samples per light source, and was rendered using the free render engine Yafray [23]. In addition, 200,000 photons were released in order to construct the Photon Map structure. With this configuration, the rendering on a single machine without optimizations took 121:17 (121 minutes and 17 seconds).

In the case of Yafrid, as we can see in Figure 6 (*Left*), the rendering time in the best case is nearly seven times better using the grid, and less than twice as good in the worst case. With these results, it is clear the importance of choosing an appropriate 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) grows, 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 aggressive optimization levels (Big or Very Big) may result in a loss of detail. For example, in Figure 7.e, the reflections on the glass are not as detailed as in Figure 7.a. The difference between the optimal render and the most aggressive optimization level (Figure 7.f) is minimal.

6 Discussion and Conclusion

The computational requirements 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 **OSCAR**-based cluster has some interesting characteristics:

- Very good throughput in the case of animations. The system divides each frame of the animation into 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 idle time (at night).
- The latency due to the file transfer is minimal (thanks to the use of a Fast Ethernet network).

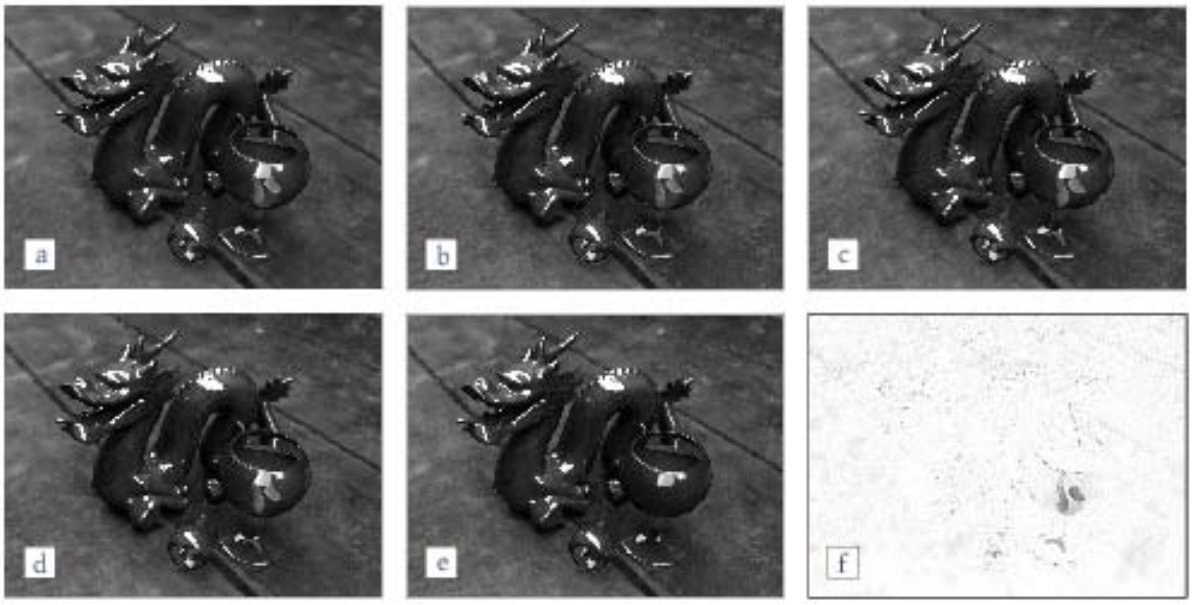

Figure 7: Result of the Rendering Using Different Optimization Levels. (a) No Optimization and Render in one Machine. (b) Very Small (c) Small (d) Normal (e) Very Big (f) Difference between (a) and (e) (the Lighter Colour, the Smaller Difference).

Otherwise, the cluster can only be used by 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 should be done to improve the Yafrid performance. Some of them were added to **MagArRO**:

- MAgArRO enables importance-driven rendering through the use of importance maps.
- It allows us to use 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 deployment 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 at [24].

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 at the Supercomputation Service (University of Castilla-La Mancha).

References

[1] D.P. Anderson, G. Fedak. The Computational and Storage Potential of Volunteer Computing. Sixth IEEE International Symposium on Cluster Computer and the Grid (CCGRID '06). p. 73-80. May 2006.

[2] I.Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. Proceedings of SIGGRAPH '04, p.777-786.

[3] A. Chalmers, T. Davis, E. Reinhard. Practical Parallel Rendering. Ed. A. K. Peters, 2002. ISBN: 1-56881-179-9.

[4] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputing Applications 15, 3(2002).

[5] T. Hachisuka. 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] J.T. Kajiya. The rendering equation. Computer Graphics 20(4): 143-150. Proceedings of SIGGRAPH '86.

[7] R.R. Kuoppa, C.A. Cruz, D. Mould. Distributed 3D Rendering System in a Multi-Agent Platform. Proceedings of the Fourth Mexican International Conference on Computer Science, 8, 2003.

[8] R. Rajagopalan, D. Goswami, S.P. Mudur. Functionality Distribution for Parallel Rendering. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), p. 18-28, April 2005.

[9] E. Reinhard, A.J. Kok, F.W. Jansen. Cost Prediction in Ray Tracing. Rendering Techniques '96, p. 41-50. Springer-Verlag, June 1996.

[10] E. Veach, L.J. Guibas. Metropolis light transport. Proceedings of SIGGRAPH '97, p. 65-76. New York, USA: ACM Press - Addison Wesley Publishing Co.

[11] T. Whitted. An improved illumination model for shaded display. Proceedings of SIGGRAPH '79, 14. New York, USA: ACM Press.

[12] M.J. Wooldridge. An introduction to multiagent systems. John Wiley & Sons, 2002. ISBN: 0-471-49691-X

[13] L.A. Zadeh. 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>.