A MultiAgent System for Physically based Rendering Optimization

Carlos Gonzalez-Morcillo[1,2], Gerhard Weiss[2], Luis Jimenez[1], David Vallejo[1], and Javier Albusac[1]

[1] Escuela Superior de Informatica,
University of Castilla-La Mancha, Spain
[2] Software Competence Center GmbH
Hagenberg, Austria
Carlos.Gonzalez@uclm.es,Gerhard.Weiss@scch.at,Luis.Jimenezo@uclm.es,
David.Vallejo@uclm.es,JavierAlonso.Albusac@uclm.es
http://oreto.inf-cr.ulcm.es - http://www.scch.at

Abstract. Physically based rendering is the process of generating a 2D image from the abstract description of a 3D Scene. Despite the development of various new techniques and algorithms, the computational requirements of generating photorealistic images still do not allow to render in real time. Moreover, the configuration of good render quality parameters is very difficult and often too complex to be done by non-expert users. This paper describes a novel approach called *MAgarRO* (standing for "*Multi-Agent AppRoach to Rendering Optimization*") which utilizes principles and techniques known from the field of multi-agent systems to optimize the rendering process. Experimental results are presented which show the benefits of *MAgarRO* -based rendering optimization.

Key words: MultiAgent, Rendering, Global Illumination, Optimization

1 Introduction

The process of constructing an image from a 3D model comprises several phases such as modelling, setting materials and textures, placing the virtual light sources, and finally rendering. Rendering algorithms take a description of geometry, materials, textures, light sources and virtual cameras as input and produce an image or a sequence of images (in the case of an animation) as output. There are different rendering algorithms – ranging from simple and fast to more complex and accurate ones – which simulate the light behavior in a precise way. Such methods are normally classified in two main categories, namely, local and global illumination algorithms. High-quality photorealistic rendering of complex scenes is one of the key goals and challenges of computer graphics. Unfortunately this process is computationally intensive and may require a huge amount of time in some cases (especially when global illumination algorithms are used), and the generation of a single high quality image may take several hours up to several days, even on fast computers. As pointed out by Kajiya [1], all rendering algorithms aim to model the light behavior over various types of surfaces and try to

solve the so-called rendering equation (which forms the mathematical basis for all rendering algorithms). Because of the huge amount of time it requires, *the rendering phase is often considered to be a crucial bottleneck in photorealistic projects*. In addition, the selection of the input parameters and variable values of the scene (number of samples per light, depth limit in ray tracing, etc.) is very complex. Typically a user of a 3D rendering engine tends to "over-optimize", that is, to choose values that increase the required rendering time considerably without affecting the perceptual quality of the resulting image.

This paper describes a novel optimization approach called *MAgarRO* based on principles, techniques and concepts known from the area of multi-agent systems. Specifically, *MAgarRO* is based on design principles of the FIPA standards (http://www.fipa.org), employs adaptation and auctioning, and utilizes expert knowledge. The key advantages of this approach are robustness, flexibility, scalability, decentralized control (autonomy of the involved agents), and the capacity to optimize locally.

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 most promising issues related to parallel and distributed rendering. This section also surveys approaches which aim at applying Artificial Intelligence methods to rendering optimization and, more specifically, it points to related work on rendering based on multi-agent technology. The next section describes *MAgarRO* in detail. Then, in the next section empirical results are shown that have been obtained for different numbers of agents and input variables. The final section offers a careful discussion and concluding remarks.

2 Related Work

There are a lot of rendering methods and algorithms with different characteristics and properties (e.g., [13, 1, 12]). Common to these algorithms is that different levels of realism of the rendering are always related in one way or another to the complexity and computation time required. Consequently, a key problem in realistic computer graphics is the time required for rendering due to the computational complexity of the related algorithms. Chalmers et al. [3] expose various research lines in the rendering optimization issues.

Optimization via Hardware. Some researchers use programmable GPUs (Graphics Processing Units) as massively parallel, powerful streaming processors than run specialized portions of code of a raytracer [6]. Other approaches are based on special-purpose hardware architectures which are designed to achieve maximum performance in a specific task [14]. These hardware-based approaches are very effective and even the costs are low if manufactured in large scale. The main problem is the lack of generality: the algorithms need to be designed specifically for each hardware architecture. Against that, *MAgarRO* works at a very high level of abstraction and runs on almost any existing rendering engine without changes.

Optimization using parallel/distributed computing. If the rendering task is divided into a number of smaller tasks (each of which solved on a separate processor), the time required to solve the full task may be reduced significantly. In order to have all processing elements fully utilized, a task scheduling strategy must be chosen. This task constitutes the elemental unit of computation of the parallel implementation approach [3], and its output is the application of the algorithm to a specified data item. There are many related approaches such as [4] which use Grid systems for rendering over the Internet. Compared to these parallel-computing approaches, *MAgarRO* uses dynamical in combination with non-centralized load balancing, and this makes *MAgarRO* more efficient especially when the number of nodes (thus the coordination overhead) increases.

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 to rendering. In fact, an estimated cost distribution is crucial in commercial rendering applications which otherwise could not be realized due to their enormous complexity. There are many approaches based on knowledge about cost distribution; a good example is [8].

Distributed Multi-Agent Optimization. The inherent distribution of multi-agent systems and their properties of intelligent interaction allow for an alternative view of rendering optimization. The work presented by Rangel-Kuoppa et al. [7] uses a JADE-based implementation of a multi-agent platform to distribute interactive rendering tasks (rasterization) across a network. The distribution of the tasks is realized in a centralized client-server style (the agents send the results of the rasterization of objects to a centralized server). 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 this communication is not knowledge-driven and no "agent-typical" mechanism such as learning and negotiation is used.

The work on stroke-based rendering (a special method of Non Realistic Rendering) proposed by Schlechtweg et al. [9] makes use of a multi-agent system for rendering artistic styles such as stippling and hatching. The environment of the agents consists of a source image and a collection of buffers. Each agent represents one stroke and executes his painting function in the environment.

2.1 Comparison to *MAgarRO*

MAgarRO , the multi-agent approach to rendering proposed in this paper, significantly differs from all related work on rendering in its unique combination of the following key features:

- **Decentralized control.** *MAgarRO* realizes rendering in a decentralized way through a group of agents coordinated by a master, where the group can be formed dynamically and most services can be easily replicated. (As regards decentralized control, *MAgarRO* follows the principle of volunteer computing [2].)

- **Higher level of abstraction.** While other approaches typically realize parallel optimization at a low level of abstraction that is specific to a particular rendering method, *MAgarRO* works with *any* rendering method. All that is required by *MAgarRO* are the input parameters of the render engine to be used.
- **Use of expert knowledge.** *MAgarRO* employs Fuzzy Set Rules and their descriptive power [11] in order to enable easy modelling of expert knowledge about rendering and the rendering process.
- **Local optimization.** Each agent involved in rendering can employ different models of expert knowledge. In this way, and by using a fine-grained decomposition approach, *MAgarRO* allows for local optimization of each rendering (sub-)task.

In combining these features, *MAgarRO* exploits and integrates some ideas from related approaches to parallel rendering. For instance, *MAgarRO* 's cost prediction map (called *Importance Map* and described below) combines prediction principles described in [5, 9] with elements of Volunteer Computing as proposed in [2] and demand driven auctioning known from agent-based task allocation.

3 The *MAgarRO* Approach

MAgarRO is a system which gets a 3D Scene as input and produces a resulting 2D image. From the point of view of the user the system works in the same way as local render engines do, but the rendering in fact is made by different agents spread over the Internet.

MAgarRO uses the ICE middleware (http://www.zeroc.com). The location service IceGrid is used to indicate in which computers the services reside. Glacier2 is used to solve the difficulties related with hostile network environments, making available agents connected through a router and a firewall.

The overall architecture of *MAgarRO* is based on the design principles of the FIPA standard. In figure 1 the general class diagram for the architecture is shown. There are some top-level general services (*Start Service*, *Agent Management System*, *Directory Facilitator* and *Agent Communication Channel*) available to all involved agents. On start-up, an agent is provided with a root service that describes or points to all the services available in the environment.

3.1 Architectural Overview

In *MAgarRO* a *StartService* offering two operations is available: an operation called *getServiceRoot* to obtain directly a description of all basic services, and another operation called *supplyBasicService* that enables the registration of a new basic service in the system. In accordance with FIPA, a basic service is defined by a unique Service Identifier (string in our case), at least one list of Transport Addresses and a description of Service Type.

The *Agent Management System* (AMS) is a general service that manages the events that occurs on the platform. This service also includes a naming service

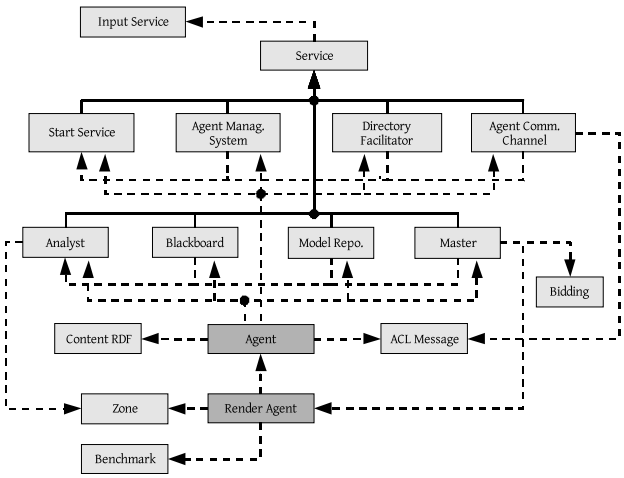

Fig. 1. *MAgarRO* general Class diagram.

for *White Pages* which allow agents to find one another. The basic functionality of the AMS is to register, to modify a subscription, to unregister agents, and to search for agents.

A basic service called *Directory Facilitator* (DF) provides *Yellow Pages* for the agents. As suggested by the FIPA standard, the operations of this service are related to the services provided by an agent, the interaction protocols, the ontologies, the content languages used, the maximum live time of registration and visibility of the agent description in DF.

Finally, *MAgarRO* includes a basic service called *Agent Communication Channel* that receives and sends messages between agents. In *MAgarRO* only the receive functionality is employed, because the send operation is implemented as part of the agents. In accordance to FIPA standard, the data structure of each message is composed of two parts: the content of the message and the envelope (with information about the receiver and the sender of the message). The Agent Communication Language used in *MAgarRO* is based on XML and uses DTD as specified in the FIPA standard.

Each agent must implement a basic set of standard operations to be able to run in the *MAgarRO* environment. This operations are suspend, terminate, resume and receive a message.

In addition to the basic FIPA services described above, *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. The environmental models processed by the agents are managed by the *Model Repository Service*. Finally, a master service called (*Master*) handles

dynamic groups of agents who cooperate by fulfilling subtasks. The Figure 2 illustrates this.

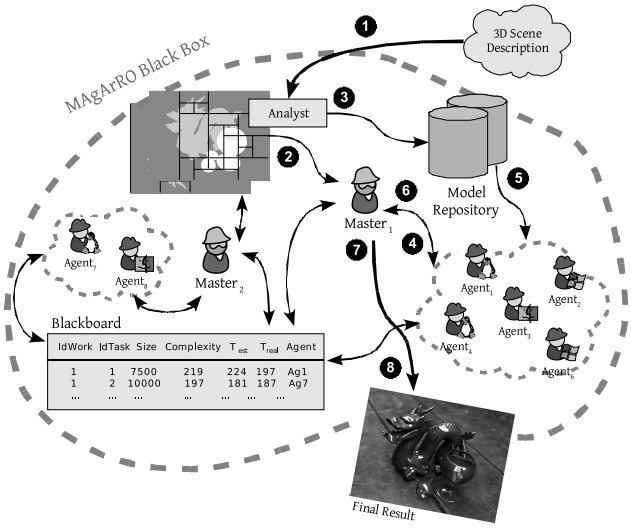

Fig. 2. General workflow and main architectural roles.

Figure 2 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 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.

3.2 Agent Subscription

As shown in Figure 1, a *Render Agent* is a specialization of a standard *Agent*, so all the functionality and requirements related with FIPA are inherited in his implementation. There are two actions that could be done by an agent: to add a subscription or to unsubscribe from a group of rendering agents. These

operations are related to a Master agent. The subscribe operation requires two parameters: the name of the agent and the *proxy* that represents the agent as a client. Using this proxy the Master could execute remote operations on the agent side. An agent can drop the association with a specific master by means of an unsubscribe operation.

The first time an agent subscribes to the system, he runs a benchmark to obtain an initial estimation of his computing capabilities. This initial value is adapted during rending in order to obtain a more accurate prediction.

3.3 Analysis of the Scene based on 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.

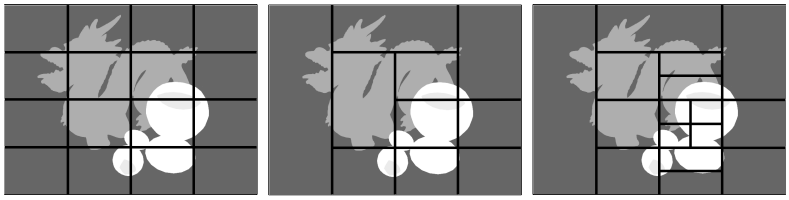

Fig. 3. Importance maps. *Left:* Blind partitioning (First Level). *Center:* Join zones with similar complexity (Second Level). *Right:* Balancing complexity/size ratio (Third Level).

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.

At the beginning, the Analyst makes a fast rasterization of the scene using an importance function to obtain a grey scale image. In this image (called *Importance Map*) the dark zones represents less complex areas and the white zones the more complex areas. In our current implementation a simple function is used (it only takes in account the recursion levels in mirror and transparent surfaces). As it is shown in Figure 3 , the glass is more complex than the dragon because it has a higher number of ray interactions. The table is less complex because it does not have any of these properties. More advanced importance functions could be used in this grey scale image generation, using perception-based rendering algorithms (based on visual attention processes) to construct the importance map [10].

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, 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 3). 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.

3.4 Rendering Process

Once the scene is available in the *Model Repository*, the *Master* assigns agents to the individual tasks identified by the Analyst. These agents, in turn, apply in parallel a technique called profiling in order to get a more accurate estimation of the complexity of each task.[1] Specifically, the agents make a low resolution render (around 5% of the final number of rays) of each task and announce on the *Blackboard* the estimated time required to do the final rendering on the blackboard.

Blackboard service The blackboard used by the agents to share their knowledge about the rendering task is implemented as a service offering a set of read and write operations. The basic blackboard data structure (as shown in Figure 2) has 7 fields labelled as follows. *IdWork* is a unique identifier for each scene, *IdZone* is a unique identifier for each task of each work, *Size* is the number of pixels of each task (width x height), *Complexity* is the estimated complexity of this task, calculated by means of the importance map, T_{est} is the Estimated Time calculated through profiling by the agents, T_{real} is the actual time required to finish a task, and *Agent* is the name of the agent who is responsible for rendering this task.

Adaptation As said above, the estimated time is represented in the same way by all agents. More precisely, each agent has an internal variable that represents his relative computational power (V_{cp}). For example, assume that $Vcp = 1$ is chosen as a reference value. If a task T_A requires 5 minutes to be done by a particular agent A who has $V_{cp} = 0.5$, he annotates on the blackboard that the time required to complete T_A is 10 minutes (i.e., the agent takes into account that he runs on a *relatively* fast machine). On the other hand, if another agent B running on a slow machine ($V_{cp} = 2$) announces that she needs 2 minutes to complete task T_B, then agent A can infer from this information that it would take him 30 seconds to complete T_B on his machine.

During run time, each agent adapts the value of his variable V_{cp} to obtain a more accurate estimation of the required processing time as follows. Whenever

[1] Profiling is a technique which traces a small number of rays in a global illumination solution and uses the time taken to compute this few rays to predict the overall computation time.

there is a difference between the estimated time T_{est} and the actual completion time T of a task, then an agent updates his internal variable V_{cp} according to

$$V_{cp} = (1 - k) \times V_{cp} + k \times (T - T_{est}) \quad (1)$$

where k a constant. Small values of k assure a smooth adaptation. (k is set to 0.1 in the experiments reported below.) This mechanisms should be improved with a more complex learning method that takes in account the historical behavior and the intrinsic characteristics of the task (type of scene, rendering method, etc...).

Auctioning At every moment during execution, all agents who are idle take part in an auction for available tasks. It is assumed that the agents try to obtain more complex tasks first. If two or more agents bid for the same task, the Master assigns it on the basis of the so called credits of these agents. The credit of an agent represents his the success and failure w.r.t. previous tasks: an agent is said to have success w.r.t. a task if he completes it in no more time than T_{est}, and otherwise he is said to fail w.r.t. this task. The amount added to, or subtracted from, an agent's credit is proportional to the time difference w.r.t. T_{est}.

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 exensibility. The output parameters (i.e., the consequent part of the rules) are configured so that the time required to complete rendering is reduced and the loss of quality is minimized. Each agent may model (or capture) different expert knowledge with a different set of fuzzy rules. In the following, the rule set we used for Pathtracing rendering is described. The output parameters of the rules are:

- **Recursion Level** [Rl], defined over the linguistic variables [15] {VS, S, N, B, VB}[2]. This parameter defines the global recursion level in raytracing (number of light bounces).
- **Light Samples** [Ls], defined over the linguistic variables {VS, S, N, B, VB}. This parameter defines the number of samples per light in the scene. The biggest, the more quality in the scene and the higher rendering time.
- **Interpolation Band Size** [Ibs], defined over the linguistic variables {VS, S, N, B, VB}. This parameter defines the size of the interpolation band in pixels, and it is used in the final composition of the image (as we will see in the next section).

The previous parameters have a strong dependency with the rendering method chosen (in this case Pathtracing). Against that, the following parameters, which are the antecedents of the rules, can be used for other rendering methods as well.

[2] The notation used for the linguistic variables is typical in some works with Fuzzy Sets. This is the correspondence of the linguistic variables: VS is *Very Small*, S is *Small*, N is *Normal*, B is *Big* and finally VB is *Very Big*.

- **Complexity** [C], defined over the linguistic variables {VS, S, N, B, VB}. This parameter represents the complexity/size ratio of the task.
- **Neighbor Difference** [Nd], defined over the linguistic variables {VS, S, N, B, VB}. This parameter represents the difference of complexity of the current task in relation to its neighbor tasks.
- **Size** [S], defined over the linguistic variables {S, N, B}. This parameter represents the size of the task in pixels (calculated as width x height).
- **Optimization Level** [Op], defined over the linguistic variables {VS, S, N, B, VB}. This parameter is selected by the user, and determines the level of optimization (more or less aggressive with initial parameters indicated by the user).

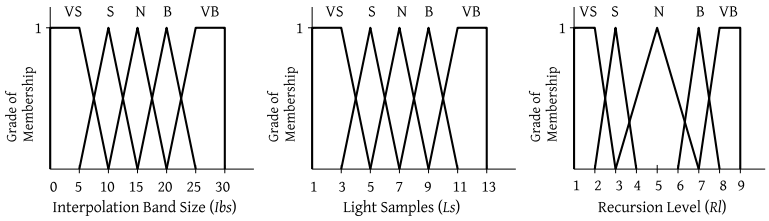

Fig. 4. Definition of the output variables.

The definition of the fuzzy sets of input variables is done dynamically in that the intervals of this sets are calculated in runtime. For example, in a highly complex scene VS is higher than VB is in a simple scene. The partition of these variables is made by linear distribution. The same occurs with other parameters like Size and Neighbor difference. In the case of the Pathtracing method, the rule set is defined as follows (only two of 28 rules are shown, all rules have been designed by an expert in PathTracing):

- R_1: **If** C is {B,VB} $\wedge$ S is B,N $\wedge$ Op is VB
  then Ls is VS $\wedge$ Rl is VS
- R_{22}: **If** Nd is VB **then** Ibs is VB

The output variables have their own fuzzy sets; we use trapezoidal functions as shown in Figure 4.

3.5 Final Result Composition

With the results generated by the different agents, the *Master* composes the final image. A critical issue w.r.t. composition is that there may be slight differences (e.g., in coloring) between the neighboring parts obtained from different agents; these differences result from the random component which PathTracing contains as a Monte Carlo based method. Figure 5 (Left) illustrates this problem. For that

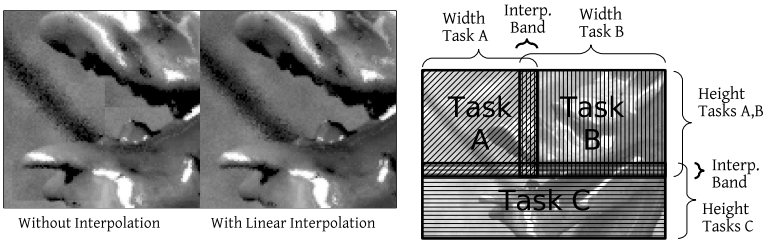

Fig. 5. Left: Without interpolation undesirable artefacts appear between neighboring parts. Linear interpolation solves this problem (at a price of slightly higher rendering costs). **Right:** Diagram of task decomposition and interpolation band situation.

reason, the *Master* smoothes the meeting faces of neighboring parts through a linear interpolation mask. More precisely, as shown in Figure 5 (Right) smoothing between neighboring parts is done in a zone called Interpolation Band.

In *MAgarRO* the size of the Interpolation Band is an output parameter of the rule set. In particular, the parameter gets a higher value if the difference between the quality of neighboring zones is important. To reduce rendering time, this parameter should be kept as small as possible to avoid unnecessary "'double work"' done by different agents. This is particularly important if the zone is very complex, as this also implies high costs for rendering the interpolation band. (In our applications the amount of time required by *MAgarRO* for interpolation was between 2% and 5% of the overall rendering time.)

4 Experimental Results

The results reported in this section have been generated with the implementation of *MAgarRO* [3]. Moreover, these results are based on the following computer and parameter setting: eight identical computers were connected to run in parallel (Pentium Intel Centrino 2 Ghz, 1GB RAM and Debian GNU/Linux); as a rendering method Pathtracing (*Yafray* 0.0.9 render engine (http://www.yafray.org)) was used; eight oversampling levels; eight recursion levels in global configuration of raytracing; and 1024 Light samples by default. The scene to be rendered contains more than 100.000 faces, 5 levels of recursion in mirror surfaces and 6 levels in transparent surfaces (the glass). With this configuration, rendering on a single machine without any optimization took 121 minutes and 17 seconds (121:17 for short, below the amount of time needed for rendering is sometimes expressed in the format *Minutes:Seconds*).

Table 1 shows the time required using different partitioning levels. These times ha have been obtained using the *N* (*Normal*) optimization level (Figure 6 Left). Using a simple first-level partitioning, a good render time can be obtained

[3] Download at (http://code.google.com/p/masyro06/) under GPL Free Software License

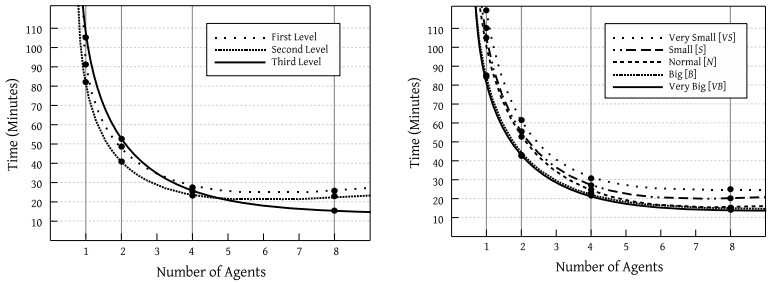

Fig. 6. Left: First/second/third level of partitioning with the N (*Normal*) optimization level. **Right:** Different optimization levels (all with third level of partitioning).

with just a few agents in comparison to third-level partitioning. The time required in the third partitioning level is larger because more partitions in areas having higher complexity (i.e., in the glasses) are needed. This higher partition level requires the use of interpolation bands and as an effect some complex parts of the image are rendered twice. For example, the rendering time with one agent is *105* minutes in the third level and *93* minutes in first level. However, when the number of agents grow, the overall performance of the system increases because the differences in the complexity of the tasks are relatively small. In first- and second-level partitioning, there are complex tasks that slow down the whole rendering process even if the number of agents is increased (the time required with four or eight agents is essentially the same). On the other hand, the third partitioning level works better with a higher number of agents.

Table 2 shows the time required to render the scene using different levels of optimization but always third-level partitioning (Figure 6 Right). By simply using a *Small* level of optimization results are obtained that are better than the results for rendering without optimization. The time required with *Very Small* optimization is exceeds the time to render the original scene. This is because additional time is required for communication and composition.

Excellent results are also obtained when only four agents are used. For instance, in the case of Normal level optimization, the time required to render the scene is just about 26 minutes (whereas the original rendering time is 120 minutes). Figure 7 shows the results of rendering obtained for different configurations.

Table 1. Different partitioning with *Normal* optimization level.

Agents	1^{st} Level	2^{nd} Level	3^{rd} Level
1	92:46	82:36	105:02
2	47:21	41:13	52:41
4	26:23	23:33	26:32
8	26:25	23:31	16:52

Table 2. Third level of partitioning with different Number of Agents and level of optimization.

Agents	VS	S	N	B	VB
1	125:02	110:50	105:02	85:50	85:06
2	62:36	55:54	52:41	42:55	42:40
4	31:10	27:11	26:32	22:50	22:40
8	23:43	20:54	16:52	16:18	15:58

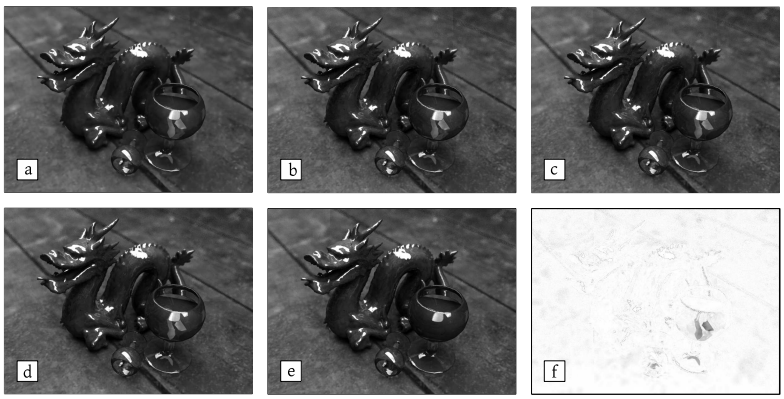

Fig. 7. Result of the rendering using different optimization levels. (a) No optimization and render in one machine. (b) *Very Small* (c) *Small* (d) *Normal* (e) *Very Big* (f) Difference between *(a)* and *(e)* (the lighter colour, the smaller difference).

As a final remark, note that optimization may result in different quality levels for different areas of the overall scene. This is because more aggressive optimization levels (i.e., Big or Very Big) may result in a loss of details. 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[4].

5 Discussion and Conclusion

The media industry is demanding high fidelity images for their 3D scenes. The computational requirements of full global illumination are such that it is practi-

[4] In this example, the difference between the optimal render (Figure 7.a) and the image obtained with the *Very Big* optimization level (see Figure 7.e) is 4.6% including the implicit noise typical to monte carlo methods (around 1.2% in this example).

cally impossible to achieve this kind of rendering in reasonable time on a single computer. *MAgarRO* has been developed in response to this challenge.

The experimental results show that *MAgarRO* achieves excellent optimization results. The use of the importance map assures an initial time estimation that minimizes the latency of the latest task. In particular, as a result of optimization *MAgarRO* achieves overall rendering times that are below the time required by one CPU divided by the number of agents. Most important, *MAgarRO* is a novel multi-agent rendering approach that offers several desirable features which together make it unique and of highest practical value. In particular:

- It is FIPA-compliant.
- Due to the use of ICE Grid middleware layer, *MAgarRO* can be used in heterogeneous hardware platforms and under different operating systems (including GNU/Linux, MacOSX, Windows, etc.) without any changes in the implementation.
- It enables importance-driven rendering through its use of importance maps.
- It employs effective auctioning and parameter adaptation, and it allows the application of expert knowledge in form of flexible fuzzy rules.
- It applies the principles of decentralized control and local optimization, and thus is scalable and very robust e.g. against hardware failures. The services are easily replicable, thus possible bottlenecks in the final deploy can be minimized.
- In presence of failures (e.g. an agent does not complete an assigned task or obtain a wrong result), it is easy to apply the typical techniques of volunteer computing systems [2].

MAgarRO is pioneering in its application of multi-agent principles to 3D realistic rendering optimization. This opens several interesting research avenues. Specifically, we think it is very promising to extend *MAgarRO* toward more sophisticated adaptation and machine learning techniques. The open technology used in the development of *MAgarRO* allow the agents to reside and run on different machines around the world. Such a Grid version is a very effective answer to the challenge of handling the enormous complexity of real-world rendering applications.

In our current work, we concentrate on two research lines. First, the combination of different rendering techniques within the *MAgarRO* framework. Due to the high abstraction level of *MAgarRO* , in princple different render engines can be combined to jointly generate an image, using complex techniques only if needed. Second, we are exploring the possibilities to equip *MAgarRO* with agent-agent real-time coordination schemes that are more flexible than auctioning.

6 Acknowledgments

This work has been funded by the Junta de Comunidades de Castilla-La Mancha under Research Project PBC06-0064.

References

1. Kajiya J. T.: The Rendering Equation. Computer Graphics 20(4): 143-150. Proceedings of SIGGRAPH 1986.
2. Anderson, D. P., Fedak, G: The computational and storage potential of volunteer computing. In Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 73-80, 2006.
3. Chalmers, A., Davis, T., Reinhard, E.: Practical Parallel Rendering. Ed. AK Peters (2002).
4. Fernandez-Sorribes, J. A., Gonzalez-Morcillo, C., Jimenez-Linares, L: Grid architecture for distributed rendering. In Proc. SIACG '06, 141-148 (2006).
5. Gillibrand, R., Debattista, K., Chalmers, A: Cost prediction maps for global illumination. In Proceedings of Theory and Practice of Computer Graphics 2005, 97-104.
6. Hachisuka, T.: GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computa tion. Addison-Wesley Professional (2005).
7. Kuoppa, R. R., Cruz, C. A., Mould, D: Distributed 3D rendering system in a multiagent platform. In Proc. ENC'03 (2003), 8.
8. Reinhard, E., Kok, A. J. F., Jansen, F. W.: Cost Prediction in Ray Tracing. In Proceedings of the Seventh Eurographics Workshop on Rendering, 41?50. New York (1996).
9. Schlechtweg, S., Germer, T., Strothotte, T: Renderbots multiagent systems for direct image generation. In Computer Graphics Forum 2005, volume 24, 137-148.
10. Sundstedt, V., Debattista, K., Longhurst, P., Chalmers, A.: Visual attention for efficient high-fidelity graphics. In Spring Conference on Computer Graphics (SCCG 2005), 162-168.
11. Tanaka, K. An introduction to fuzzy logic for practical applications. Springer (1998).
12. Veach, E., Guibas, L. J: Metropolis light transport. In SIGGRAPH '97, 65-76. New York (1997).
13. Whitted, T.: An improved illumination model for shaded display. In SIGGRAPH '79, 14. New York (1979).
14. Woop, S., Schmittler, J., Slusallek, P: Rpu: a programmable ray processing unit for realtime ray tracing. In ACM SIGGRAPH 2005, 434-444. New York.
15. Zadeh, L. A. The concept of a linguistic variable and its applications to approximate reasoning. part i, ii, iii. Information Science (1975).