A Multi-Agent Approach to Distributed Rendering Optimization

Carlos Gonzalez-Morcillo*
Escuela Superior de Informatica
University of Castilla-La Mancha, Spain

Gerhard Weiss
Software Competence Center GmbH
Hagenberg, Austria

Luis Jimenez and **David Vallejo**
Escuela Superior de Informatica
University of Castilla-La Mancha, Spain

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.

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. 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, and the generation of a single high quality image may take several hours up to several days on fast computers. 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 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. 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.

Related Work

There are a lot of rendering methods and algorithms with different characteristics and properties (e.g., (Kajiya 1986; Veach & Guibas 1997)). Chalmers et al. (Chalmers, Davis, & Reinhard 2002) 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 (Hachisuka 2005). Other approaches are based on special-purpose hardware architectures which are designed to achieve maximum performance in a specific task (Woop, Schmittler, & Slusallek 2005). 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.

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 global time may be reduced significantly. In order to have all processing elements fully utilized, a task scheduling strategy must be chosen. There are many related approaches such as (Fernandez-Sorribes, Gonzalez-Morcillo, & Jimenez-Linares 2006) which use Grid systems on the Internet.

*Also research at Software Competence Center GmbH in Hagenberg, Austria

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. There are many approaches based on knowledge about cost distribution; a good example is (Reinhard, Kok, & Jansen 1996).

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. (Kuoppa, Cruz, & Mould 2003) uses a JADE-based implementation of a multi-agent platform to distribute interactive rendering tasks (rasterization) across 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 this communication is not knowledge-driven and no "agent-typical" mechanism such as learning and negotiation is used.

Comparison to *MAgarRO*

MAgarRO 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.
- **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 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 (Gillibrand, Debattista, & Chalmers 2005; Schlechtweg, Germer, & Strothotte 2005) with elements of Volunteer Computing as proposed in (Anderson & Fedak 2006) and demand driven auctioning known from agent-based task allocation.

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 (http://www.zeroc.com) middleware. 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.

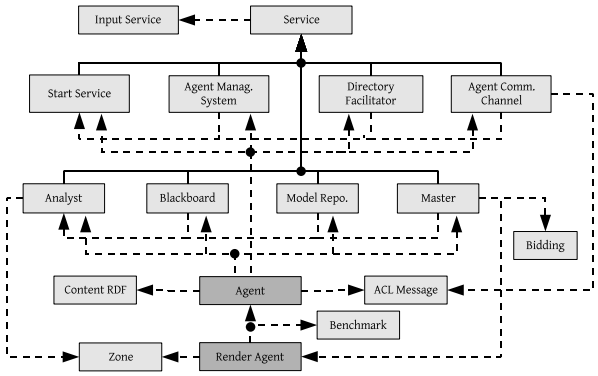

Figure 1: *MAgarRO* general Class diagram.

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 startup, an agent is provided with a root service that describes or points to all the services available in the environment.

Architectural Overview

In *MAgarRO* 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 for *White Pages* which allow agents to find one another. 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 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.

Figure 2 also illustrates the basic workflow in *MAgarRO* (the circled numbers in this figure represent the follow-

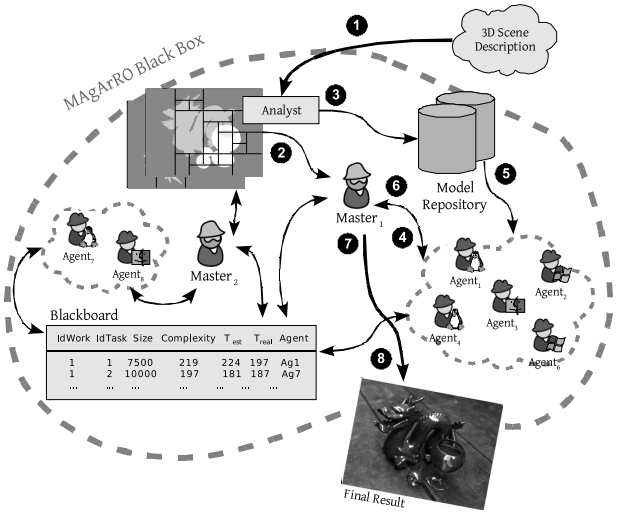

Figure 2: General workflow and main architectural roles.

ing 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.

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. 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.

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 partitioning.

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. As it is shown in Figure 3, the glass is more complex than the dragon because it has a higher number of ray interactions.

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.

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, *IdTask* 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}). 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 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

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

where k a constant. Small values of k assure a smooth adaptation. (k is set to 0.1 in the experiments reported below.)

[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.

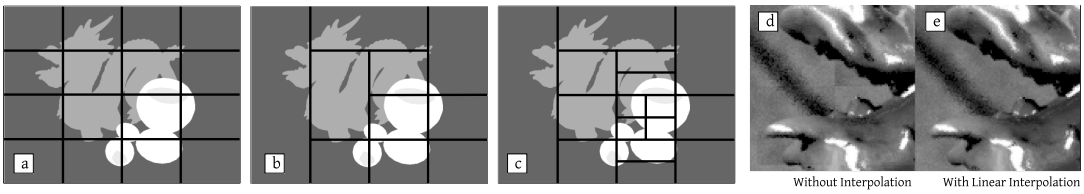

Figure 3: **Importance maps (a-c)**. *(a)* Blind partitioning (First Level). *(b)* Join zones with similar complexity (Second Level). *(c)* Balancing complexity/size ratio (Third Level). **Artifacts (d-e)**. (d) Without interpolation undesirable artefacts appear between neighboring parts. (e) Linear interpolation solves this problem (at a price of slightly higher rendering costs).

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 the success and failure w.r.t. previous tasks.

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. 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 (Zadeh 1975) {VS, S, N, B, VB}[2]. This parameter defines the global recursion level in raytracing (number of light bounces).
- **Light Samples** [Ls], defined over {VS, S, N, B, VB}. Number of samples per light in the scene.
- **Interpolation Band Size** [Ibs], defined over {VS, S, N, B, VB}. Size of the interpolation band in pixels. 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.

- **Complexity** [C], defined over {VS, S, N, B, VB}. Complexity/size ratio of the task.
- **Neighbor Difference** [Nd], defined over {VS, S, N, B, VB}. Difference of complexity of the current task in relation to its neighbor tasks.
- **Size** [S], defined over {S, N, B}. Size of the task in pixels (width x height).
- **Optimization Level** [Op], defined over {VS, S, N, B, VB}. This parameter is selected by the user.

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.

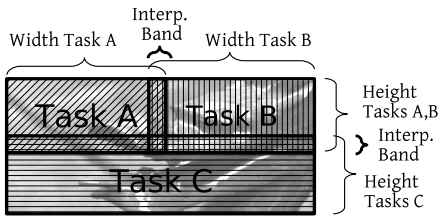

Figure 5: Diagram of task decomposition and interpolation band situation.

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 3.d illustrates this problem. For that reason, the *Master* smoothes the meeting faces of neighboring parts through a linear interpolation mask called Interpolation Band (Figures 3.e, 5).

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

[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*.

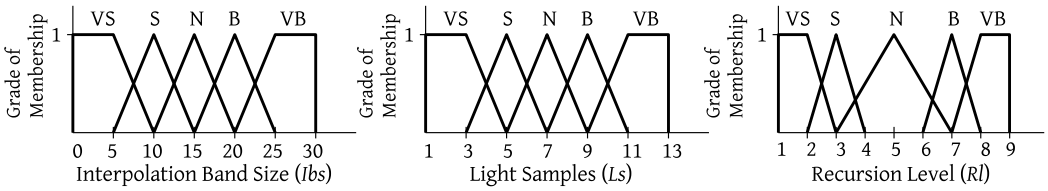

Figure 4: Definition of the output variables.

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.)

Experimental Results

The results reported in this section have been generated with the implementation of *MAgarRO* which we have made available for download at (http://code.google.com/p/masyro06/) under GPL Free Software License. Moreover, these results are based on the following computer and parameter setting: eight identical computers were connected to run in parallel (Centrino 2 Ghz, 1GB RAM running Debian); as a rendering method Pathtracing (*Yafray* render engine (http://www.yafray.org)) was used; eight recursion levels in global configuration of raytracing; and 1024 Light samples. 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 [3].

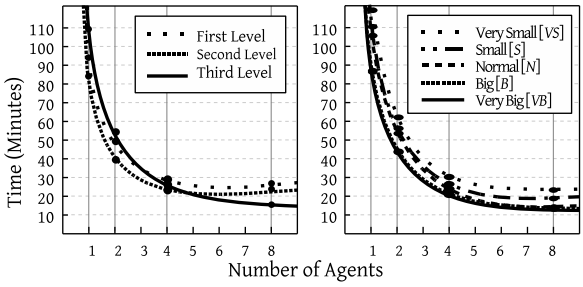

Figure 6: **Left:** First/second/third level of partitioning with the N (*Normal*) optimization level. **Right:** Different optimization levels (all with third level of partitioning).

Table 1 shows the time required using different partitioning levels. These times 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 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

[3] 121:17 for short, below the amount of time needed for rendering is sometimes expressed in the format *Minutes:Seconds*.

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

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 exceeds the time to render the original scene. This is because additional time is required for communication and composition.

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.c, 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.c) is minimal[4].

[4] In this example, the difference between the optimal render (Figure 7.a) and the image obtained with the *Very Big* optimization

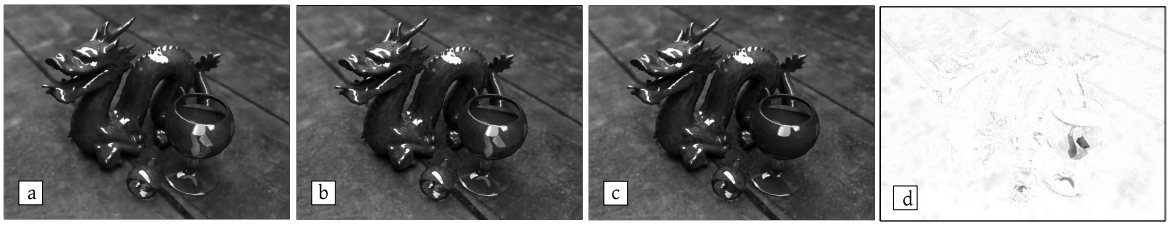

Figure 7: Result of the rendering using different optimization levels. (a) No optimization and render in one machine. (b) *Normal* (c) *Very Big* (d) Difference between *(a)* and *(c)* (the lighter colour, the smaller difference).

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

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.
- *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 (is scalable). The services are easily replicable (possible bottlenecks in the final deploy can be minimized).
- In presence of failures, it is easy to apply the typical techniques of volunteer computing systems.

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. 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.

level (see Figure 7.d) is 4.6% including the implicit noise typical to monte carlo methods (around 1.2% in this example).

Acknowledgments

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

References

Anderson, D. P., and Fedak, G. 2006. The computational and storage potential of volunteer computing. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 73–80.

Chalmers, A.; Davis, T.; and Reinhard, E. 2002. *Practical Parallel Rendering*. AK Peters Ltd.

Fernandez-Sorribes, J. A.; Gonzalez-Morcillo, C.; and Jimenez-Linares, L. 2006. Grid architecture for distributed rendering. In *Proc. SIACG '06*, 141–148.

Gillibrand, R.; Debattista, K.; and Chalmers, A. 2005. Cost prediction maps for global illumination. In *TPCG 05*, 97–104. Eurographics.

Hachisuka, T. 2005. *GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional.

Kajiya, J. T. 1986. The rendering equation. *Computer Graphics* 20(4):143–150. Proc. of SIGGRAPH'86.

Kuoppa, R. R.; Cruz, C. A.; and Mould, D. 2003. Distributed 3d rendering system in a multi-agent platform. In *Proc. ENC'03*, 8.

Reinhard, E.; Kok, A.; and Jansen, F. W. 1996. Cost Prediction in Ray Tracing. In *Proceedings 7th Eurographics Workshop on Rendering*, 41–50.

Schlechtweg, S.; Germer, T.; and Strothotte, T. 2005. Renderbots – multiagent systems for direct image generation. In *Computer Graphics Forum*, volume 24, 137–148.

Veach, E., and Guibas, L. J. 1997. Metropolis light transport. In *SIGGRAPH '97*, 65–76. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Woop, S.; Schmittler, J.; and Slusallek, P. 2005. Rpu: a programmable ray processing unit for realtime ray tracing. In *Proc. SIGGRAPH '05*, 434–444.

Zadeh, L. A. 1975. The concept of a linguistic variable and its applications to approximate reasoning. part i, ii, iii. *Information Science*.