This article was downloaded by: *[Morcillo, Carlos Gonzalez]*
On: *13 April 2010*
Access details: *Access Details: [subscription number 921256342]*
Publisher *Taylor & Francis*
Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK

Applied Artificial Intelligence
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713191765

A MULTIAGENT ARCHITECTURE FOR 3D RENDERING OPTIMIZATION

Carlos Gonzalez-Morcillo [ab]; Gerhard Weiss [b]; David Vallejo [a]; Luis Jimenez-Linares [a];Jose Jesus Castro-Schez [a]
[a] Escuela Superior de Informatica, Paseo de la Universidad, Ciudad Real, Spain [b] Software Competence Center GmbH, Hagenberg, Austria

Online publication date: 12 April 2010

To cite this Article Gonzalez-Morcillo, Carlos , Weiss, Gerhard , Vallejo, David , Jimenez-Linares, Luis andCastro-Schez, Jose Jesus(2010) 'A MULTIAGENT ARCHITECTURE FOR 3D RENDERING OPTIMIZATION', Applied Artificial Intelligence, 24: 4, 313 — 349
To link to this Article: DOI: 10.1080/08839511003715212
URL: http://dx.doi.org/10.1080/08839511003715212

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: http://www.informaworld.com/terms-and-conditions-of-access.pdf

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.

Applied Artificial Intelligence, 24:313–349
Copyright © 2010 Taylor & Francis Group, LLC
ISSN: 0883-9514 print/1087-6545 online
DOI: 10.1080/08839511003715212

Taylor & Francis
Taylor & Francis Group

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

A MULTIAGENT ARCHITECTURE FOR 3D RENDERING OPTIMIZATION

Carlos Gonzalez-Morcillo[1,2], Gerhard Weiss[2], David Vallejo[1], Luis Jimenez-Linares[1], and Jose Jesus Castro-Schez[1]

[1]*Escuela Superior de Informatica, Paseo de la Universidad, Ciudad Real, Spain*
[2]*Software Competence Center GmbH, Hagenberg, Austria*

□ *Rendering is the process of generating a 2D image from the abstract description of a 3D scene. In spite of the development of new techniques and algorithms, the computational requirements of photorealistic rendering are huge so that it is not possible to render them in real time. In addition, the adequate configuration of rendering quality parameters is very difficult to be done by inexpert users, and they are usually set higher than in fact are needed. This article presents an architecture called MAgArRO to optimize the rendering process in a distributed, noncentralized way through a multiagent solution, by making use of expert knowledge or previous jobs to reduce the final rendering. Experimental results prove that this novel approach offers a promising research line to optimize the rendering of photorealistic images.*

INTRODUCTION

The process of generating a 2D image comprises several phases such as modeling, setting materials and textures, placing virtual light sources, and finally rendering (Kerlow 2004). Rendering algorithms take a description of geometry, materials, textures, light sources, and the virtual camera as input to produce an image or a sequence of images (in the case of animations) as output. There are different rendering algorithms ranging from simple and fast to complex and accurate that simulate the light behavior in a precise way (Pharr and Humphreys 2004). These methods are normally classified into two main categories (Goral et al. 1984): local and global illumination algorithms.

This work has been funded by the Junta de Comunidades de Castilla-La Mancha under Research Projects PII2I09-0052-3440 and PII1C09-0137-6488. Special thanks to Javier Galan for his indoor scene used in this work for testing the system.

Address correspondence to Carlos Gonzalez-Morcillo, Escuela Superior de Informatica, Paseo de la Universidad 4, 13071 Ciudad Real, Spain. E-mail: Carlos.Gonzalez@uclm.es

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

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 when the rendering technique simulates global illumination issues. Global illumination algorithms are known for their unpredictable data accesses and their computational complexity (Jensen 2004). As pointed out by Kajiya (1986), all rendering algorithms aim at modeling the light behavior over various types of surfaces and try to solve the rendering equation, which forms the mathematical basis for all rendering algorithms. Depending on the rendering method and the scene characteristics, the generation of a single high-quality image may take several hours (or even days!).

Because of the huge amount of time required, the rendering phase is often considered as a bottleneck in photorealistic projects in which one image may need hours of rendering in a modern workstation. For instance, Pixar's famous animation movie *"Cars"* was estimated to spent 2300 CPU years by one of the technology manager of the studio. In other words, a single computer would have to run over 2300 years to perform the rendering of all the frames that compose the movie. In fact, the first rendering tests took 10 hours for a single frame. If the running time of the film is 116 minutes and the frame rate is 24 frames per second, that is, each second of the film requires 24 generated images, then rendering becomes an important problem. In addition, the adequate configuration of input parameters and variables values of the scene (number of samples per light, number of photons, depth limit in ray tracing, etc.) is very complex. Commonly, the user of the 3D rendering engine chooses very high values that do not affect to the perceptual quality of the resulting image. Unfortunately, this fact increases even more the rendering time of the scene without actually improving the final result.

To face these challenges, we introduce an optimization approach based on principles, techniques, and concepts known from the area of multiagent systems. The key advantages of this approach are robustness, flexibility, good scalability, decentralized control (autonomy), and the capacity of performing local optimizations thanks to the use of expert knowledge. Through this novel and pioneering approach we aim to reach three main goals: i) to reduce the time spent in the rendering of photorealistic images by means of a distributed approach, ii) to optimize critical rendering parameters through the acquisition of expert knowledge or learning based on previous experience by rendering agents, and iii) to provide a multiagent framework to advance the state of the art on distributed rendering. The approach devised in this article has been extensively evaluated with good results in complex scenes.

The rest of the article is structured as follows. We first discuss the alternatives and research lines in rendering optimization. We focus our effort on one of the most promising issues related with parallel and

distributed rendering. In this section we also analyze previous work that applies artificial intelligence methods for rendering optimization and, more specifically, the related work made in multiagent systems. The architecture of MAgArRO (multiagent architecture for rendering optimization) is described in depth next. The functional prototype of MAgArRO implements the core of a multiagent system based on the design principles of the FIPA standard (FIPA n.d.a). The empirical results obtained with this prototype, using a different number of agents and input variables, are exposed in the next section. Finally, we resume the main contributions of our work and conclude with some ideas about future research lines.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

RELATED WORK

There are various rendering methods with different characteristics and properties that make each algorithm more appropriated for diverse application domains. One of the most famous algorithm is the ray tracing algorithm (Whitted and Holmdel 1980). An extension of this method proposed by Cook et al. (1984) and called distributed ray tracing uses Monte Carlo techniques for integration as the classical version of the algorithm. There are some tricks like ambient occlusion, such as in Zhukov et al. (1998) and Hurley (2005), that simulate the effect of global illumination. Radiosity techniques, originally introduced by Goral et al. (1984), are based on the principle of light exchange between surfaces. According to the Monte Carlo integration method, path tracing was the first method that supported all kinds of light transport paths and produced correct images (from the point of view of the light transport). This technique was proposed by Kajiya (1986) in the same article in which the rendering equation was introduced. One technique widely used to avoid noise in path tracing is the use of irradiance cache, proposed by Ward et al. (1988). Bidirectional path tracing was proposed by Lafortune et al. (1993) and consists in generating ray paths from the light sources and from the virtual camera. The metropolis light transport algorithm was proposed by Veach and Guibas (1997) and works well in hard lighting situations (like small holes in a surface where the light enters or in caustics simulations). This algorithm, like the bidirectional path tracing, also makes use of the Monte Carlo approach to solve global illumination problems in two phases.

There are many techniques based on the previous algorithms, but it is not the purpose of this work to make an analysis of rendering methods in depth. The different levels of realism of rendering methods are related to complexity and computation time spent. Even more, with the new acquisition techniques, high-resolution screens (monitors, TV, cinema), and the look for hyperrealism in many application areas, the rendering

time (and the requisites of resolution, quality, and complexity of scenes) will keep on growing up.

As previously discussed, one of the key problems in photorealistic computer graphics is the time spent in rendering due to the unpredictable data accesses and the high computational complexity of related algorithms. Several alternatives minimize the computation time (Chalmers et al. 2002). In the next subsections some options to optimize this task are described.

Optimizations via Hardware

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Modern graphics processing units (GPUs), currently integrated in most personal computers, can be used as massively parallel and powerful streaming processors that run concrete portions of the code of a ray tracer. The use of programmable GPUs outperforms the standard workstation CPUs by a factor of approximately seven (Buck et al. 2004). This characteristic, together with the low cost and the high processing speed of modern GPUs (which are doubling their performance every 6 months), currently represents a tendency that consists in using this unit as a parallel specific processor in some rendering issues. The use of the CPU in conjunction with the GPU requires new paradigms and alternatives to traditional architectures. For example, the architectural configurations proposed by Rajagopalan et al. (2005) demonstrate the use of a GPU to work on real-time rendering of complex data sets that demand complex computations or the load balanced algorithm to render unstructured grids with hardware and software (Weiler and Ertl 2001). There are some rendering engines designed to be used with GPU acceleration, such as the Parthenon Renderer (Hachisuka 2005), or the Gelato Heath (2008).

Special purpose hardware architectures can also be built to achieve the maximum performance in specific tasks. PURE uses the fully programmable ray tracing hardware architecture RPU (Woop et al. 2005), which hosts dedicated hardware for spacial division structures. McEwan et al. (2007) proposed modifications to the basic ray tracing algorithm by targeting reconfigurable hardware devices and giving an estimation for the hardware needed.

Ray tracing can be exploited in the future as an alternative to the z-buffer algorithm for interactive use by directly implementing it in GPUs (Shirley et al. 2008). The impact in image synthesis community will be very important.

These alternatives are very effective in time optimization. However, the lack of generality of most of them is a significant problem. In fact, these algorithms have to be specifically designed for each hardware architecture, and some optimizations are even made with the help of specialized chips.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Optimizations Using Parallel Computing

Another option to optimize the rendering involves the use of parallel or distributed systems. If the main problem is divided into a number of smaller problems (each of which is solved by an individual processor), the time spent to solve the whole problem is reduced. Although generally this is true, all processing elements must be fully utilized and a task scheduling strategy must be adopted. Within this context the task forms the elemental unit of computation of the parallel implementation (Chalmers et al. 2002), and its output is the application of the algorithm to a specified data item. For example, in ray tracing one task may involve computing the value of some set of pixels at the image plane.

One common alternative for solving a problem on a parallel system consists of using a domain decomposition approach, which runs an algorithm on different data items (tasks) in parallel. The domain decomposition can be done by using a data-driven or a demand-driven approach. In the data-driven model (Figure 1a and b) the tasks are assigned to the processing units before starting to compute. In the other alternative, the demand-driven model (Figure 1c), the tasks are dynamically allocated to the processing units as they become idle. This is done by implementing a pool of available tasks, and the processing units request the tasks to be done. The parallel architecture may be designed to keep all the processing units fully utilized. This is not trivial because the proper nature of rendering implies that different zones of the image plane may have a different complexity due to some characteristics like the geometry of the object, material, texture, and lights and shadows. The differences in the computational effort associated with the tasks increase the probability for the processing elements to become idle.

To avoid this effect, the tasks must be allocated by taking into account the complexity of each one and by assigning them to the processors in a balanced way. In Figure 1b, the tasks have been assigned to the processors

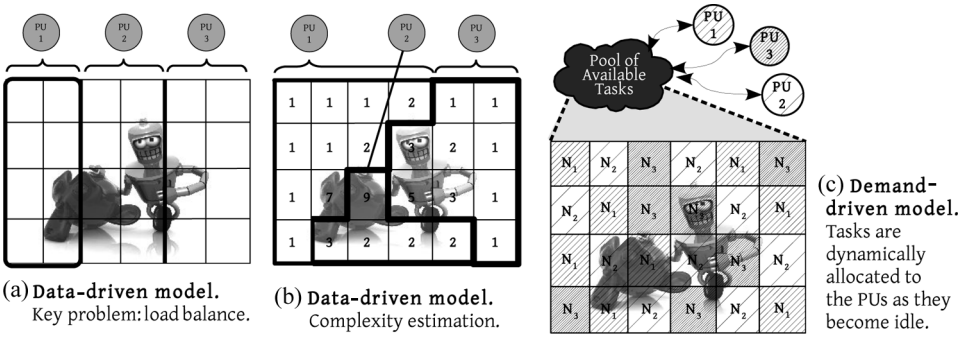

FIGURE 1 Data- and demand-driven models for the allocation of the tasks (PU = processing unit).

in sets of 18 *complexity units*, that is, the sum of the complexities of the work units that compose a task is equal to 18. In this example, processor 2 performs a lower number of tasks than processor 1, but the more complicated ones.

In the demand-driven computational model, the tasks are dynamically allocated to the processing units as they become idle. When a processing unit obtains the result of a task, it demands for the next task of the pool (Figure 1c). The dynamic allocation of tasks ensures that although there are available tasks, the processing units will be working on them. This approach facilitates the load balancing scheme so that there are only load imbalances with the latest tasks. For instance, if the last task for PU_1 is very simple and the last one for PU_2 is very complex, PU_1 will become idle before PU_2 finishes its task. This problem can be avoided by sorting the tasks depending on their complexity. In this way, the more complex tasks should be allocated before the less complex ones. This approach minimizes the idle time with the latest tasks.

The more relevant group of distributed and parallel rendering systems (in the image synthesis area; there are a lot of works in the interactive graphics field that are out of the scope of our work) is formed by dedicated clusters and rendering farms. Some 3D animation companies use their rendering farms in their own productions, and some of them offer rendering services via Internet. An user of these rendering services can make use of the dedicated cluster that the company owns. Depending on how to do the division of tasks, we distinguish between fine-grained systems, in which each image is divided into small parts that are sent to a processor to be independently rendered, or coarse-grained (in case of animations), in which each frame of an animation is entirely rendered by a computer.

For example, Jevans (1989) used a dynamic load balancing (demand-driven) scheme with spacial partitioning based on voxels. Other approaches, like Ris and Arques (1994), use a static load balancing scheme on a network with sequential workstations and parallel computers. Other approaches oriented to render animations, like Stober et al. (1988) and Martino and Köhling (1992), incorporate coarse-level dynamic load balancing (demand-driven) schemes and distribute individual frames of an animation to computers connected to the system.

There are many approaches based on the completely redesign of rendering algorithms to achieve high efficiency in distributed environments. Although global illumination algorithms are becoming more and more complex, the effort needed to analyze and adapt the code to a parallel system is also very high, and the result usually affects the design of the original algorithm so that it is very hard to maintain. Examples of these approaches can be found in Snell and Gustafson (1997), Kato and Saito (2002), and Günther et al. (2004). Other alternatives, like that proposed by Kipfer and Slusallek (1999), transparently introduce

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

distributed logic in the code of existing object-oriented solutions (by using the CORBA [Henning and Vinoski 1999] middleware).

New approaches of distributed rendering use the grid approach (Foster et al. 2002) to allocate tasks among a large number of heterogeneous computers connected to the Internet (Patoli et al. 2008; Gooding et al. 2006). Volunteer computing is a research line that uses the idle time of processors (Fernandez-Sorribes et al. 2006). This kind of system uses computers connected to the Internet, volunteered by their owners, as a source of computing power for rendering 3D scenes by using different rendering engines (Anderson et al. 2005; Anderson and Fedak 2006).

The main advantage of these alternatives is the high-efficiency obtained when using existing computers. There is no need of specialized hardware, and in many organizations a cluster of available workstations is enough. One of the key problems is related with the effective load balancing and the difficult management when the number of the nodes connected to the server grows up.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Distributed Multiagent Optimizations

The distributed approach is a good option to optimize the rendering process. In spite of the fact that the optimization made in this way can improve the global rendering time in a huge factor, there are some key quality parameters selected by the user that play a critical role in this time. These input parameters are, for example, the number of samples per pixel, the recursion level in ray tracing, the number of photons in the photon mapping technique, and so on. Usually, inexpert users choose values higher than needed, without improving the perceptible quality of the final image but with an important increase of the rendering time. To avoid this situation, expert knowledge could be modeled to automatically select a good configuration of the input parameters. Therefore we have a good starting point based on distributed systems, but some expert knowledge could be added to obtain better results. The inherent distribution of multiagent systems and the interactions between intelligent agents configure a promising alternative for rendering optimization.

There are few previous attempts on the use of this approach for optimizing the rendering process. The work presented by Rangel-Kuoppa et al. (2003) used a multiagent platform (a JADE-based implementation) to distribute interactive rendering tasks (pure rasterization) across a network. The distribution of the task is based on a centralized client-server approach, in which the agents send the results of the rasterization of objects to a centralized server. Although using a multiagent architecture, there is no specific use of the multiagent technology. The use of the JADE framework (Bellifemine et al. 2007) is only for communication issues between nodes, but there is no knowledge, learning, or auctions.

The authors used a multiagent platform to reduce the development time by using a tested tool. Finally, they conclude that adopting a multiagent architecture for interactive rendering is not a good option, due to the excessive time consumed in communication and the low performance as a result of using a Java implementation.

The work in stroke-based rendering (a special method of nonrealistic rendering) proposed by Schlechtweg et al. (2005) makes use of a multiagent system for rendering artistic styles such as stippling and hatching. The environment of the agents consists in a source image and a collection of buffers. Each agent represents one stroke and executes its painting function in the environment. These simple agents (called render bots) execute three actions: simulation, to control the physical behavior of the agent; movement, to translate the agent to other position in the environment; and painting, to generate one stroke in the final image.

Our approach is based on a multiagent architecture (Weiss 1999) that allows us to distribute the rendering tasks between the agents registered with the multiagent system. We devise different agents depending on the role that play within the Multi-Agent System (MAS): agents responsible for managing the submission of rendering works and the correct composition of partial results (*master agent*), agents responsible for performing an analysis of input scenes to divide the whole work into different tasks by balancing their complexity (*analyst agent*), and agents responsible for rendering these tasks by making use of knowledge base acquired through a knowledge acquisition subsystem (*rendering agent*). These rendering agents are also capable of participating in auctions when they are notified about a new existing work. The auction mechanism takes into account the number of credits and the historical behavior of each rendering agent to distribute the tasks with the aim of reducing the global rendering time. The prototype of this specific distributed rendering has been deployed over a previously developed general-purpose FIPA-compliant multiagent framework to empirically demonstrate how this novel approach based on expert knowledge optimises the rendering process.

MAgArRO APPROACH

This section gives a general description of the approach adopted in this work. First a multi-agent system designed according to the FIPA committee (FIPA n.d.a) has been developed to provide the basis for agent-based solutions in different application domains. The aim of this approach lies in providing a common set of management services shared by any specific multiagent system devised to solve a particular problem. In our case this problem is distributed rendering. Thanks to this approximation, we are able to reuse this set of basic services for any problem that requires an agent-based solution, obtaining an architectural base that can be adopted

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

to develop specific agents for a particular domain. Next, we briefly describe the basic services of a multi-agent system designed according to FIPA specifications.

The **Agent Management System** (**AMS**) is a general service that manages the events that occur in the platform and controls the state of each agent. This service also offers the white pages service that allows agents to discover one another. The basic functionality of the AMS is to register agents, modify subscriptions, unregister agents, and search for agents with specific characteristics.

The **Directory Facilitator** (**DF**) implements the yellow pages service for agents. The operations are basically the same that in the AMS, but the internal data structure and the behavior are different. The implementation of this service follows the indications of the FIPA standard, by modeling the parameters related to the services provided by an agent, interaction protocols, ontologies, known content languages, maximum live time of registration, and visibility of the agent description in the DF. This service is persistent and is automatically activated on demand. The agents are also notified about new events thanks to the subscription mechanism provided by this service.

The **Agent Communication Channel** (**ACC**) receives and sends messages between agents. According to the 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).

From this general-purpose MAS a set of agents and services specifically designed for the problem of distributed rendering is deployed. From an abstract point of view, the MAgArRO system can be viewed as a black box that takes a 3D scene as input and produces the resulting 2D rendered image. Inside this black box there is an undefined number of agents that cooperate to perform the distributed rendering and a set of well-defined services that give support to specific tasks. Figure 2 graphically shows this set of rendering agents and services. Next, we briefly describe the functionality of each one them:

- **Master**: This agent is responsible for managing a group of rendering agents. The main abilities of the master consist in supervising the rendering of a previously submitted work and composing the partial results provided by the rendering agents which take part in such work.
- **Analyst**: This agent analyzes the 3D scene to be rendered to study its complexity. When this is done the 3D scene is divided into a number of tasks (work units) with a similar complexity. In this way a master can orchestrate the distributed rendering of these tasks between a set of rendering agents.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

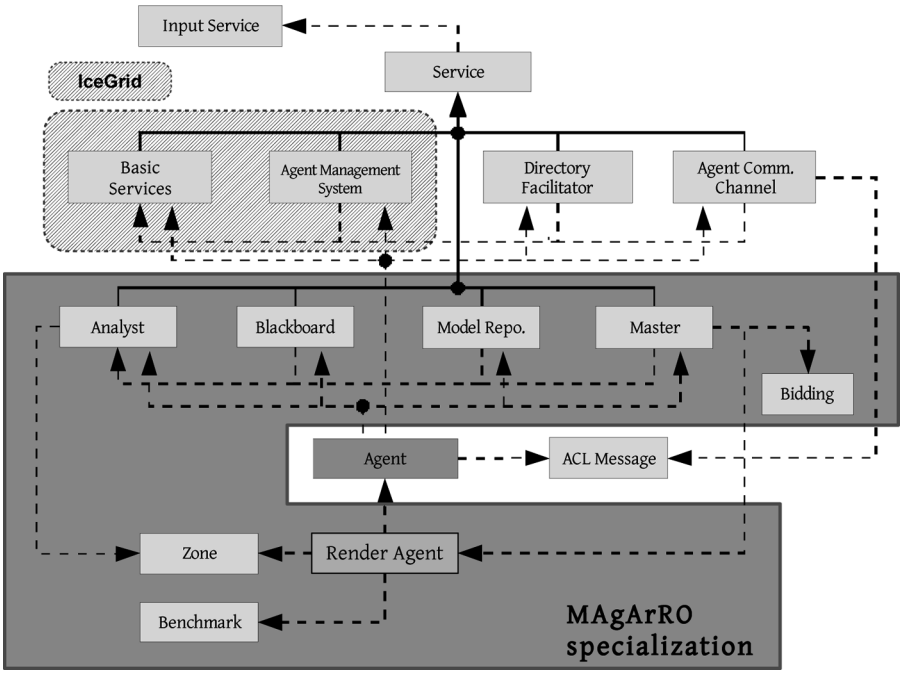

FIGURE 2 MAgArRO general class diagram (the framed area represents the agents and services that compose MAgArRo).

- **Rendering Agent**: This agent is capable of performing the rendering of tasks by applying a knowledge base acquired through the knowledge acquisition subsystem or tool (Castro-Schez et al. 2004) or learned from previous experiences using machine learning algorithms (Castro et al. 1999) to optimize this process by adjusting relevant rendering parameters. As is shown in Figure 2, a Rendering Agent is a specialization of the FIPA standard agent so that all the functionality and requirements specified by FIPA are inherited by the rendering agent.
- **Model Repository**: This service gives support for storing the 3D models rendered. Basically, this repository allows the rendering agents to download the whole 3D scene submitted by the user.
- **Blackboard**: This service maintains the data structures used to write/read the progress of rendering a 3D scene by a set of rendering agents. The blackboard provides the rendering agents with the communication mechanism needed to perform the distributed rendering in a parallel way.

The basic workflow of MAgArRO is shown in Figure 3, where the circled numbers represent the following steps:

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

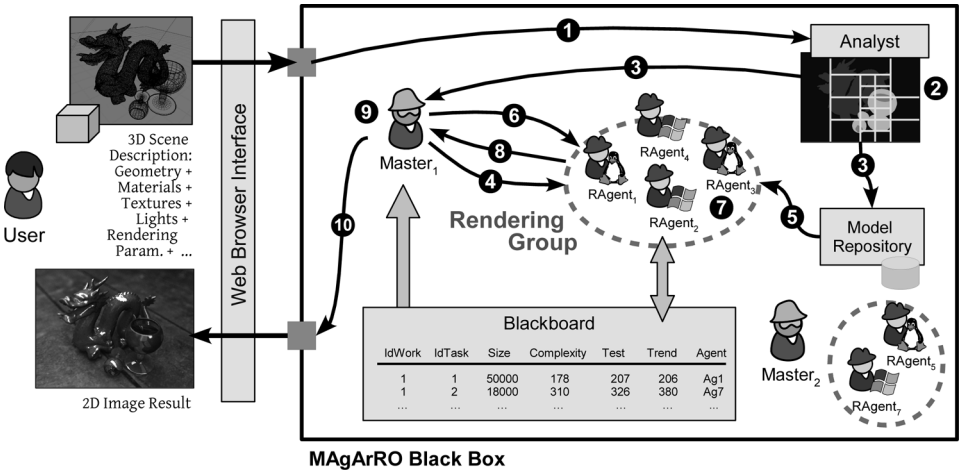

FIGURE 3 General workflow and main architectural roles of MAgArRO.

1. The system receives a new work (the user submits such work to the *Analyst* by means of a public interface). In the current development version, a web interface is provided for the user to easily submit a work. Previously, the rendering agents were subscribed to one of the *Masters* of the system (see below). This subscription can be done in any moment so that the available agents are dynamically managed.
2. The *Analyst* makes a fast analysis of the scene by dividing the work into a number of tasks that share a similar complexity (see below).
3. The *Analyst* uploads the model to the *Model Repository* and notifies the existing work to a *Master*.
4. The *Master* notifies the new work to some of the *Rendering Agents* (possibly running on different computers) that are available.
5. Each *Rendering Agent* downloads the 3D model from the repository and the auction process managed by the *Master* begins.
6. The *Master* assigns the work units to each Rendering Agent.
7. The Rendering Agents start to render such tasks.
8. When a *Rendering Agent* has rendered a task, the result of such work units is sent to the *Master* and the *Rendering Agent* bids for another work unit.
9. The final result is composed by the *Master* from the partial results previously submitted by the *Rendering Agents*.
10. The *Master* sends the final rendered image to the user.

Next, we introduce the formalization of MAgArRO (M'):

$$M' = \langle S, A, R, \mathcal{M}, B, I \rangle$$

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

where

1. S is the input scene to be rendered.
2. A is the *Analyst* agent, which can be define as follows:

$$A = \langle CS, ES, O \rangle$$

being CS an estimation of the complexity of S and ES a strategy used to divide S into O, which represents a set of scene partitions with similar complexity. We refer each one of these partitions as a *working unit*. That is, $O = \{wu_1, \ldots, wu_n\}$ with each wu_i being a work unit.
3. R is the *Model Repository* service that stores the set of partitions O generated by the analyst A.
4. $\mathcal{M}$ is defined as follows:

$$\mathcal{M} = \{(m_i, RA_i) \mid i = 1 \ldots n\}$$

where in each pair (m_i, RA_i), m_i represents a *Master* agent and RA_i is a finite set of *Rendering Agents* working for him, i.e., $RA_i = \{ra_{i1}, ra_{i2}, \ldots, ra_{im}\}$.

- A Master service m_i assigns working units $wu_j \in O$ to each rendering agent ra_{ij} in RA_i and analyses the partial results provided by each ra_{ij} to compose the final image I. A master m_j is defined as follows:

$$m_j = \langle IDm_j, KBS_j \rangle$$

where

  - IDm_j is the master's identifier, which is unique in the system execution domain.
  - KBS_j is the master's knowledge base, which consists of a blending method and a set of rules to determine its parameters.

- RA_i is a finite set of Rendering Agents, $RA_i = \{ra_{i1}, ra_{i2}, \ldots, ra_{im}\}$. Each Rendering Agent $ra_{ij} \in RA_i$ is defined as follows:

$$ra_{ij} = \langle IDR_j, KBS_j, Cr_j, H_j, CP_j, ST_j \rangle$$

where

- $IDra_j$ is the rendering agent's identifier, which is unique in the system execution domain.
- KBS_j is the agent's knowledge base, which consists of rendering methods and sets of rules to optimize the rendering parameters depending on the assigned task.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

- Cr_j represents the agent's number of credits. This element represents the agent's historical behavior ($Cr \in \mathbb{N}$) used to check if the rendering agent performs the rendering tasks in a time shorter or similar than the previously estimated by the analyst.
- H_j is the agent's recent history. It is a set of boolean values that represent *success* or *failure* in the last working units assigned (it was carried out in the right time or not).
- CP_j is the agent's computational capacity. This parameter depends on the computer where the rendering agent runs ($CP_j \in \mathbb{R}^+$).
- ST_j is the internal agent's state, which can take values from the set {*Estimating, Bidding, Rendering, Resting, Finishing*}.

5. B is the Blackboard service. This component maintains useful information about the input scene S and the rendering process, such as the work units assigned to each ra_{ij} in a concrete moment.
6. I is the resulting 2D image rendered.

In the next section, we describe in detail each one of the elements of the formalization and how we have addressed the problem of distributed rendering.

MAgArRO MULTIAGENT ARCHITECTURE

Analyzing the Input 3D Scene

As we discussed previously, it is very convenient to have an estimation of the complexity of the different work units that compose the input scene to be rendered previously submitted by the user. In our architecture this service is provided by the *Analyst Agent* ($A = \langle CS, ES, O \rangle$). The main goal of this agent is to achieve a good load balancing of the work units that compose the input scene, that is, the main objective in this partitioning process is to obtain work units with a similar complexity to avoid the delay in the final time caused by too complex tasks. This analysis is carried out in a fast way independently of the rendering algorithm used. For each submitted 3D scene (Figure 4a), the Analyst generates an importance map as shown in Figure 4b to determine what parts of the scene are complex and what parts are not (see Gillibrand et al. 2005, 2006 for similar approaches).

In the beginning the Analyst performs a fast rendering of the scene by means of an importance function to obtain a gray scale image (CS) to estimate the complexity of the model (see the corresponding sequence diagram in Figure 5). In this image (called Importance Map), the dark zones represent the less complex areas and the white zones the more complex ones. Currently, a function that only takes into account the

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

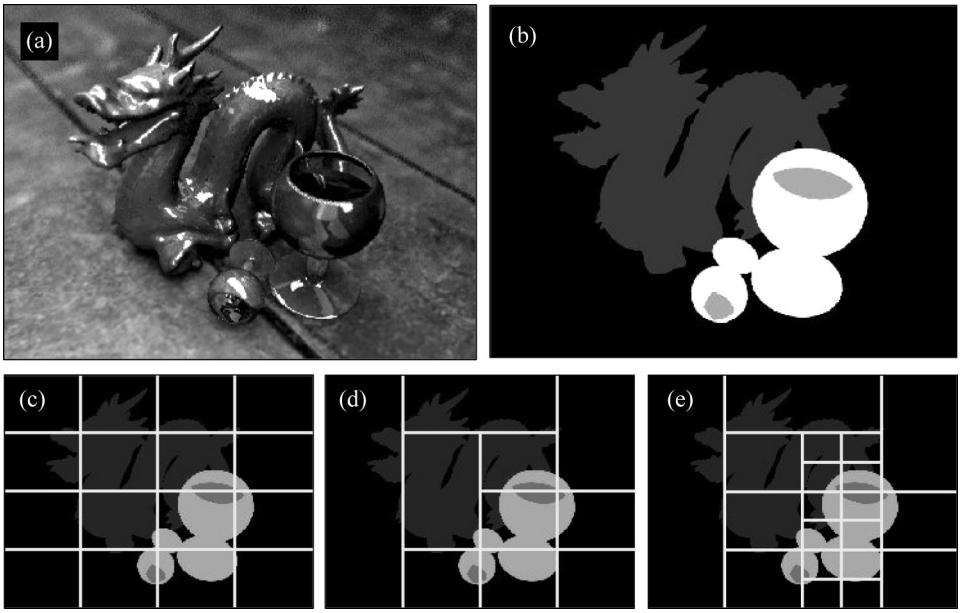

FIGURE 4 Importance maps. (a) Resulting 2D image obtained after rendering the dragon 3D scene. (b) Importance map. (c) Blind partitioning (First level). (d) Joined zones with similar complexity (Second level). (e) Balancing complexity/size ratio (Third level).

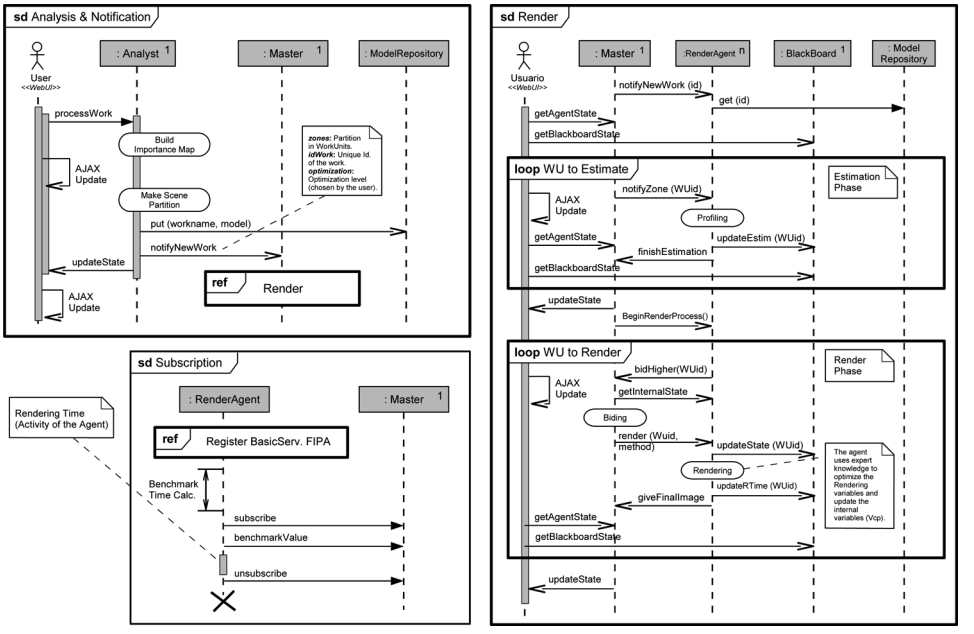

FIGURE 5 Sequence diagrams in AUML notation. *Top Left*: Analysis and Notification of new work protocol. *Bottom Left*: Agent subscription protocol. *Right*: Rendering protocol.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

recursion levels in mirror and transparent surfaces is used. As shown in Figure 4b, the glass is more complex that the dragon because it has mirror and transparency properties with a higher number of ray interactions than other objects. The table is less complex because it does not have any of these properties.

Once the importance map is generated, the next step consists in applying the strategy (*ES*) developed to divide the input scene *S* into *O*, that is, the set of partitions or tasks with a similar complexity performed later by the rendering agents. Afterward, the analyst is able to upload to the model repository the whole scene that contains the 3D model and to notify this set of tasks to a master m_i (see top left of Figure 5). The strategy *ES* allows to made the task division in different levels so that each partitioning level is made by using the information of the previous level.

In the first level the partition is made by taking into account the minimum size and the maximum complexity of each zone. With these two parameters, the Analyst makes a recursive division of the work units (Figure 4c). Algorithm 1 describes the first level of partitioning. In this

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Algorithm 1 Blind Recursive Division BlindDivision (Figure 4c)

```
Require: wu_n // n^th work unit
// Calculate the mean m and the typical deviation d
s ← 0, n ← 0, L(wu_n) ← φ
for i = wu_n(x_1) to wu_n(x_2) do
  for j = wu_n(y_1) to wu_n(y_2) do
    s ← wu_n.pixel(i, j)
    n ← n + 1
  end for
end for
m ← s/n
s ← 0
for i = wu_n(x_1) to wu_n(x_2) do
  for j = wu_n(y_1) to wu_n(y_2) do
    s ← (wu_n.pixel(i, j) − m^2)
  end for
end for
d ← √(s/n)
// Recursive partitioning of the work unit wu_n
if d >MAXDEV and wu_n.size >MINSIZE then
    x_1 ← wu_n(x_1), x_2 ← wu_n(x_2), y_1 ← wu_n(y_1), y_2 ← wu_n(y_2)
    BlindDivision(WU(x_1, y_1, x_1 + (x_2 − x_1)/2, y_1 + (y_2 − y_1)/2))
    BlindDivision(WU(x_1 + (x_2 − x_1)/2, y_1, x_2, y_1 + (y_2 − y_1)/2))
    BlindDivision(WU(x_1, y_1 + (y_2 − y_1)/2, x_1 + (x_2 − x_1)/2, y_2))
    BlindDivision(WU(x_1 + (x_2 − x_1)/2, y_1 + (y_2 − y_1)/2, x_2, y_2))
else
    L(wu_n) ← L(wu_n) + wu_n // Add wu_n to list of work units
     (stop recursion)
end if
```

Algorithm 2 Joining of wu_n of Similar Complexity Joining (Figure 4d)

Require: List of wu_n ($L(wu_n)$ from algorithm 1 (BlindDivision)
  $c \leftarrow$ **true** // c indicates if there is change
  while c **do**
    $c \leftarrow$ **false**
    for wu_i in $L(wu_n)$ **do**
      for wu_j in $L(wu_n)$ **do**
        if $|wu_i.m - wu_j.m| \langle$MAXDIFF and Adj$(wu_i, wu_j)$ **then**
          $wu_k \leftarrow$ Join (wu_i, wu_j)
          $L(wu_n) \leftarrow L(wu_n) - wu_i - wu_j + wu_k$
        end if
      end for
    end for
  end while

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

algorithm the pair $[x_1, y_1]$ represents the upper left corner of the work unit wu_n, the pair $[x_2, y_2]$ represents the bottom right corner of wu_n, the function *pixel(i,j)* returns the value of each pixel of wu_n, *MAXDEV* is the maximal value allowed for the typical deviation and *MINSIZE* is the minimal size (width × height) allowed for wu_n.

In the second level of partitioning (see Algorithm 2), the neighbor work units with similar complexity are joined. In this algorithm *MAXDIFF* is the biggest difference value allowed between the value of wu_n. The *Adj* function indicates if the related wu_n are neighbors. The *Join* function returns a new wu_n as a result of the combination of the two wu_n passed as parameters.

Finally, in the third level of partitioning, the Analyst tries to obtain a balanced division with almost the same complexity/size ratio of each work unit (see Algorithm 3). In this method, *Lc* stores one list of the complexity/size ratio of each wu_n, m stores the mean of this ratios, and d the typical deviation. In the main loop, for each element of *Lc* it is necessary to make an horizontal or vertical partition.

By means of this final approximation, we try to obtain the set of tasks O with a similar complexity so that the rendering time does not differ among work units. As we discuss in the next section, the quality of this partitioning is very related with the final rendering time. It is important to remark that because of the unpredictable nature of ray tracing methods, this is only an approximation of the complexity of the work unit.

After having performed this initial estimation with a certain partitioning level, the analyst updates the blackboard with the relevant information that will be used by both a master m_i and a set of rendering agents RA_i. This information comprises the unique identifier of the work (*IdWork*), each one of the work unit identifier (wu_n) in which the whole project was divided, the size in pixels (*Size*) of each task, and

Algorithm 3 Balancing of wu_n (Complexity/Size Ratio) Balancing (Figure 4e)

Require: List of wu_n ($L(wu_n)$ from Algorithm 2 (Joining)
  for wu_i in $L(wu_n)$ **do**
    $Lc \leftarrow \text{complex}(wu_i)$ / $\text{size}(wu_i)$
  end for
  $m \leftarrow \frac{\sum_{i=0}^{n} Lc_i}{n}$
  $d \leftarrow \sqrt{\frac{\sum_{i=0}^{n} (Lc_i - m)^2}{n}}$
  $Lz \leftarrow \phi$
  for i in length(Lc) **do**
    if $m \leq Lc[i]$ **then**
      $Lz \leftarrow$ Horizontal partitioning of $Lc[i]$
      if $Lc[i] \geq m\times$ DIVFACTOR **then**
        $Lz \leftarrow$ Vertical partitioning of $Lc[i]$
      end if
      $L(wu_n) \leftarrow L(wu_n) - L(wu_n)_i + Lz$
    end if
  end for

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

the complexity (*Comp*) of each task. In this way the rest of agents of the distributed rendering platform are able to know the division previously carried out by the analyst agent.

Forming Rendering Teams

Once the Analyst has performed the complexity analysis of the input scene and before starting the distributed rendering process, we describe how the *Master Agent* m_i carries out the subscription management of the rendering agents responsible for rendering the input scene S. Basically, there exists a relation between a master agent m_i and a finite set of rendering agents RA_i. This association configures a rendering team in which the master m_i coordinates the rendering process and the rendering agents support the work of the distributed rendering process. As previously described, the architecture allows different rendering teams, that is, pairs (m_i, RA_i), to coexist within the same multiagent system. The relationship between a master m_i and a set of rendering agents RA_i is established by means of a subscription (Figure 5).

When a Rendering Agent $ra_{ij} \in RA_i$ subscribes to a master m_i through a unique identifier $IDra_j$, it previously runs a benchmark to get an initial estimation of its computing capabilities CP_j, that is, the computing power of a rendering agent ra_{ij} depends on the power of the processor integrated into the computer on which the agent runs (in fact, rendering time

essentially depends on the performance of the microprocessor). This CPU test is necessary to have an estimation of the time that the rendering of each task requires. This initial value is modified in runtime to obtain more precise predictions. The approximation obtained with the benchmark is only a first estimation.

Rendering Works

Notification of a New Work

When a new work submission S is notified to a master m_i, this agent organizes the available agents RA_i to complete the work (see the AUML diagram of Figure 5, right to appreciate the interactions between the master m_i and the set of rendering agents RA_i). First, m_i sends a message to RA_i about the notification of a new work. The *Master* knows the division into work units made by the *Analyst*; then, each ra_{ij} uses a *profiling* technique to estimate in a more accurate way the complexity of the set of tasks assigned by m_i. The idea is to distribute the estimation of tasks in which the input scene S was divided between RA_i. To do that, these agents perform a low resolution rendering (10% of the final number of rays) of each wu_i and annotate in the *Blackboard* the estimated time (T_E) required to perform the final rendering. This estimated time is calculated in a common time format for the agents of the architecture by using the computational capacity of each agent (CP_j), as seen below.

Blackboard Service

There is a common place, a blackboard structure, where rendering agents and masters share their knowledge about scenes in a blackboard structure. This component provides a service with a set of read/write operations related with the blackboard data structure, which has 11 fields labelled as follows (Table 1 shows a representative subset of the fields):

- *IdWork*: Identifier of the work. Each rendering project has an unique identifier.
- *wu*: Identifier of the work unit of the rendering project.
- *Size*: Size of the *wu* in pixels (width × height).
- *Complexity*: Complexity of the *wu*. It depends on the importance map.
- *Agent*: Identifier of the rendering agent assigned to this w_u.
- *State*: Current state of the *wu*. It can be *Waiting* to be done, rendering (*Working*) or finished (*Done*).
- T_E: Estimated rendering time for this *wu*.
- R_T: Real rendering time spent for this *wu* by using the final quality and resolution settings.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

TABLE 1 Blackboard Status After Rendering the Dragon Model (Figure 4) With Path Tracing and Normal Optimization Level

WU	Size	Comp	TE	RT	Ibs	Ls	Rl
0	76,800	2	1:19	0:47	3	7	5
1	38,400	19	1:13	0:55	3	7	5
5	38,400	0	1:23	0:40	3	7	5
11	19,200	34	1:53	5:15	10	9	7
13	19,200	36	1:39	2:04	14	9	7
15	19,200	1	1:02	0:23	15	9	7
16	19,200	67	2:01	3:41	3	9	7
17	19,200	65	2:01	3:17	4	9	7
20	4800	70	1:19	0:50	3	9	7
21	4800	56	1:15	0:41	6	9	7
22	4800	99	1:29	2:29	7	9	7
23	4800	175	2:09	4:49	13	9	7
26	4800	144	1:35	2:45	12	9	6
27	4800	230	1:39	4:44	18	10	7
28	4800	176	1:14	2:51	5	9	7
29	4800	238	1:13	2:09	3	9	7
32	4800	199	1:15	8:54	24	10	7
33	4800	245	1:12	4:34	26	9	7
34	4800	9	0:51	1:32	10	9	7
35	4800	13	0:50	0:57	12	9	7

- *Ibs*: Interpolation band size chosen by the rendering agent using its knowledge base.
- *Ls*: Number of light samples chosen by the rendering agent using its knowledge base.
- *Rl*: Recursion level in mirror and transparent surfaces (for ray tracing based methods) recommended by the rendering agent using its knowledge base.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Simple Adaptation

T_E is calculated in a common format for all rendering agents. Each ra_{ij} has an internal variable that represents its computational power (CP_j). Initially, this value is calculated by running a benchmark.

Next, we describe a possible scenario. For example, we take as the reference value $CP = 1$. If one task $t_{ra_{i1}}$ required 5 minutes to be done by a particular ra_{i1} that has $CP_1 = 0.5$, it annotates in the blackboard that the time required to do $t_{ra_{i1}}$ is 10 minutes (because this agent is running on a very fast computer). On the other hand, if another agent ra_{i2}, which is running in an old computer ($CP_2 = 2$), then ra_{i2} estimates that the task $t_{ra_{i2}}$ will require 2 minutes in such computer, then it writes on the blackboard the time in the common format, i.e., 1 minute. If ra_{i1} reads the blackboard, it knows that $t_{ra_{i2}}$ will require only 30 seconds to be done in its computer.

Through this approach we are able to estimate rendering times to achieve a good load balancing scheme by using a demand-driven (based on dynamic auctions) model in which the more complex tasks are performed in first place.

During the execution of the system, each ra_{ij} modifies the value of this variable (CP_j) to have a more accurate representation according to the common time. There is a very basic mechanism that implements a method of simple adaptation. When an agent ra_{ij} completes a task with an estimated time T_E in a time T, the internal variable CP_j is updated by means of a lineal adaptation equation:

$$CP_j = (1 - k) \times CP_j + k \times (T - T_E) \tag{1}$$

Being k a constant that in the current implementation and for most cases represents a value closed to 0.1 and ensures a soft adaptation. This mechanism should ideally be improved with a more complex learning method that takes into account the historical behavior H_A and the intrinsic characteristics of the task (type of scene, rendering method, etc.).

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Auctioning

When a set of rendering agents RA_i attached to a master m_i has completed the initial estimation of tasks in which an input scene S was divided, m_i notifies to the agents RA_i about the beginning of the auction. This mechanism has been devised to reach two main goals:

1. To distribute the tasks between RA_i so that the more complex work units are first rendered. The idea consists in reducing the global rendering time by trying to match the moment in which all the work units are done.
2. To propose a mechanism that assigns the more complex work units to the rendering agents with the highest computational capabilities.

Therefore RA_i try to obtain the more complex work units first to avoid these complex tasks to delay the simple ones. If two or more agents bid for the same work unit, then m_i assigns it to one of them by taking into account two factors:

- **The number of credits of the rendering agent** ra_{ij} (Cr_j). This parameter represents the successes and failures of ra_{ij} in previous tasks. ra_{ij} has success in one work unit if it is finished in a time less or equal than T_E for that task. In other case, ra_{ij} increases the number of failures. The amount of credits added or subtracted to ra_{ij} is proportional to the difference between the real rendering time T_R and the estimated rendering time T_E (see details in Algorithm 4).

Algorithm 4 Biding and Assignment of Credits

```
for ra_ij in L_RA_i(finishing) do
  L_b ← ra_ij.bid
end for
for b_i in L_b | i = 1..n do
  if b_i.wu ≠ b_j.wu | ∀j = 1..i − 1, i + 1..n then
    b_i.ra ← b_i.wu
  else
    b_i.val ← Σ_{m=1}^{H_S} b.ra.H + b.ra.Cr
    for b_k in L_b | k = 1..n do
      if b_i.wu = B_k.wu then
        b_k.val ← Σ_{m=1}^{H_S} b.ra.H + b.ra.Cr
        if b_k.val > b_i.val then
          b_k.ra ← b_i.wu
        else
          b_i.ra ← b_i.wu
        end if
      end if
    end for
  end if
end for
```

- **The historical behavior** (H_j). This element represents the list of latest successes and failures accumulated by the agent ra_{ij}. It consists of a set of boolean values that represent *success* or *failure* in the last working units assigned. This is used to weight the more recent activity of the agent to assign new tasks. This historical vector works as a frame of size H_S.

The auction protocol of MAgArRO is an adaptation of the FIPA contract-net protocol (FIPA n.d.b). An idle agent ra_{ij} basically can bid for a wu_k, and the master m_i decides about which one is finally assigned to it. When ra_{ij} is subscribed to m_i, its historical vector is initialized to 0 $(H_j \leftarrow 0 \mid i = 0..H_S)$ and the number or credits Cr_j is set to the initial value $Cr_j = 10$.

In Algorithm 4 the first "for loop" represents the time spent by m_i in waiting for the bidding of the idle agents (stored in L_b, the List of bids). In the second loop, $b.wu$ represents the identifier of the work unit of the bid, $b.ar_{ij}$ is the Agent that made the bid, $b.ar_{ij}.H_j$ is the historical vector of the agent and $b.ar_{ij}.Cr_j$ is the number of credits of the agent. If more than one agent bid for the same wu_k, m_i evaluates all the bids and assigns the work unit wu_k to the best candidate, that is, the one with the best recent historic.

When ra_{ij} finishes the rendering process, if the rendering time spent T_R is similar to T_E, then the number of credits Cr_j is increased by two. In other case, Cr_j is decreased by one.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Use of Knowledge-Based Systems

Each rendering agent ra_{ij} models its knowledge by means of a Knowledge-Based System KBS_j, which is composed of a set of rendering techniques:

$$KBS_j = \{est_1, est_2, \ldots, est_m\}$$

where each est_i is defined as $est_i = \langle Rm_i, KB_i \rangle$, being Rm_i a rendering method and KB_i the knowledge used to configure the method parameters in an efficient way depending on the scene characteristics. In other words each rendering method Rm_i makes use of a set of parameters (Y_i) as input which determines the behavior of Rm_i. The adequate values of these parameters can be established depending on certain characteristics of the scene (V_i) so that it is possible to establish the relationships (R_i) between them with a certain precision or uncertainty. To facilitate and make intuitive the model and description of the relevant rendering parameters Y_i and the scene characteristics V_i, we used Fuzzy Logic (Zadeh 1999).

An expert could provide us with a set of rules R_i that associates V_i to Y_i or it could be learned using machine learning techniques. In other words, the expert could define a set of rules R_i to learn the function $V_i \rightarrow Y_i$. To deal with the existing uncertainty, each variable (from V_i and Y_i) is given as a linguistic variable defined over its own definition domain DDV, which represents the set of values that can take. Next, we focus on the particular case of the path tracing rendering method. In this example, the rendering parameters Y_i to be determined are as follows:

- **Recursion Level** $[Rl]$: This parameter defines the global recursion level in ray tracing (number of light bounces).
- **Light Samples** $[Ls]$: This parameter defines the number of samples per light in the scene. Larger values of this parameter involves a higher quality in the resulting image and more rendering time T_R spent.
- **Interpolation Band Size** $[Ibs]$: This parameter defines the size of the interpolation band in pixels, and it is used in the final composition of the image (as we see in the next section).

On the other hand, the scene variables V_i that are useful to determine Y_i are as follows:

- **Complexity** $[C]$: This parameter represents the complexity/size ratio of the work unit.
- **Neighbor Difference** $[Nd]$: This parameter represents the difference of complexity of the current work unit in relation with its neighbor work units.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

- **Size** $[S]$: this parameter represents the size of the work unit measured in pixels (calculated as $width \times height$).
- **Optimization Level** $[Op]$: This parameter is selected by the user, and it determines the optimization level (more or less aggressive depending on the initial parameters customised by the user).

The knowledge base KB_i for the path tracing case is as follows:

$$KB_i = \langle V_i, Y_i, DDV, R_i \rangle$$

where

- $V_i = \{C, Nd, S, Op\}$
- $Y_i = \{Ibs, Ls, Rl\}$
- $DDV = \{DDV_C, DDV_{Nd}, DDV_S, DDV_{Op}, DDV_{Ibs}, DDV_{Ls}, DDV_{Rl}\}$ where each DDV_i is defined as follows:

– $DDV_C = \{VerySmall(VS), Small(S), Normal(N), Big(B), VeryBig(VB)\}$
– $DDV_{Nd} = \{VerySmall(VS), Small(S), Normal(N), Big(B), VeryBig(VB)\}$
– $DDV_S = \{Small(S), Normal(N), Big(B)\}$
– $DDV_{Op} = \{VerySmall(VS), Small(S), Normal(N), Big(B), VeryBig(VB)\}$
– $DDV_{Ibs} = \{VerySmall(VS), Small(S), Normal(N), Big(B), VeryBig(VB)\}$ (Figure 6)
– $DDV_{Ls} = \{VerySmall(VS), Small(S), Normal(N), Big(B), VeryBig(VB)\}$ (Figure 6)
– $DDV_{Rl} = \{VerySmall(VS), Small(S), Normal(N), Big(B), VeryBig(VB)\}$ (Figure 6)
– R_i (Table 2)

The definition of the fuzzy sets of input variables is dynamically made; the intervals of this sets are calculated in runtime. For example, in a high complex scene, the definition of VS (*Very Small*) should be higher than the

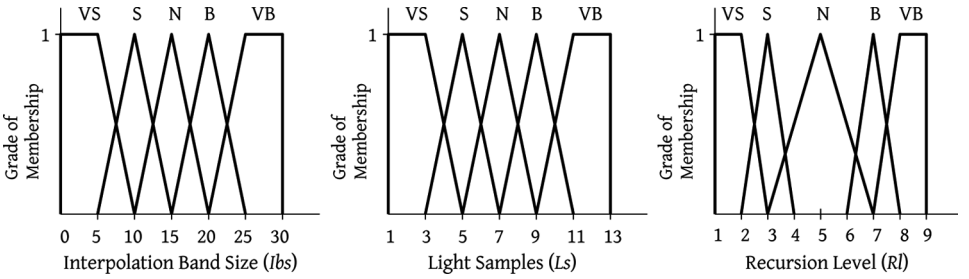

FIGURE 6 Definition of the output variables

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

TABLE 2 Rules for the Optimization of a PathTracer

R	Antecedent	Consequent
R_1:	If C is {B,VB} $\wedge$ S is {B,N} $\wedge$ Op is {VB}	$\Rightarrow$ Ls is {VS} $\wedge$ Rl is {VS}
R_2:	If C is {N} $\wedge$ S is {B,N} $\wedge$ Op is {VB}	$\Rightarrow$ Ls is {VS} $\wedge$ Rl is {VS}
R_3:	If C is {S,VS} $\wedge$ S is {B,N} $\wedge$ Op is {VB}	$\Rightarrow$ Ls is {S} $\wedge$ Rl is {S}
R_4:	If S is {S} $\wedge$ Op is VB	$\Rightarrow$ Ls is {S} $\wedge$ Rl is {N}
R_5:	If C is {B,VB} $\wedge$ S is {B,N} $\wedge$ Op is {B}	$\Rightarrow$ Ls is {S} $\wedge$ Rl is {VS}
R_6:	If C is {N} $\wedge$ S is {B,N} $\wedge$ Op is {B}	$\Rightarrow$ Ls is {S} $\wedge$ Rl is {VS}
R_7:	If C is {S,VS} $\wedge$ S is {B,N} $\wedge$ Op is {B}	$\Rightarrow$ Ls is {N} $\wedge$ Rl is {S}
R_8:	If S is {S} $\wedge$ Op is B	$\Rightarrow$ Ls is {N} $\wedge$ Rl is {N}
R_9:	If C is {B,VB} $\wedge$ S is {B,N} $\wedge$ Op is {N}	$\Rightarrow$ Ls is {N} $\wedge$ Rl is {S}
R_{10}:	If C is {N} $\wedge$ S is {B,N} $\wedge$ Op is {N}	$\Rightarrow$ Ls is {N} $\wedge$ Rl is {S}
R_{11}:	If C is {S,VS} $\wedge$ S is {B,N} $\wedge$ Op is {N}	$\Rightarrow$ Ls is {N} $\wedge$ Rl is {N}
R_{12}:	If S is {S} $\wedge$ Op is {N}	$\Rightarrow$ Ls is {B} $\wedge$ Rl is {B}
R_{13}:	If C is {B,VB} $\wedge$ S is {B,N} $\wedge$ Op is {S}	$\Rightarrow$ Ls is {B} $\wedge$ Rl is {N}
R_{14}:	If C is {N} $\wedge$ S is {B,N} $\wedge$ Op is {S}	$\Rightarrow$ Ls is {B} $\wedge$ Rl is {N}
R_{15}:	If C is {S,VS} $\wedge$ S is {B,N} $\wedge$ Op is {S}	$\Rightarrow$ Ls is {B} $\wedge$ Rl is {B}
R_{16}:	If S is {S} $\wedge$ Op is {S}	$\Rightarrow$ Ls is {VB} $\wedge$ Rl is {B}
R_{17}:	If C is {B,VB} $\wedge$ S is {B,N} $\wedge$ Op is {VS}	$\Rightarrow$ Ls is {B} $\wedge$ Rl is {N}
R_{18}:	If C is {N} $\wedge$ S is {B,N} $\wedge$ Op is {VS}	$\Rightarrow$ Ls is {VB} $\wedge$ Rl is {B}
R_{19}:	If C is {S,VS} $\wedge$ S is {B,N} $\wedge$ Op is {VS}	$\Rightarrow$ Ls is {VB} $\wedge$ Rl is {VB}
R_{20}:	If S is {S} $\wedge$ Op is {VS}	$\Rightarrow$ Ls is {VB} $\wedge$ Rl is {VB}
R_{21}:	If C is {VB,B} $\wedge$ Nd is {VB}	$\Rightarrow$ Ls is {VB}
R_{22}:	If Nd is {VB}	$\Rightarrow$ Ibs is {VB}
R_{23}:	If C is {VB,B} $\wedge$ Nd is {B}	$\Rightarrow$ Ls is {VB}
R_{24}:	If Nd is {B}	$\Rightarrow$ Ibs is {B}
R_{25}:	If C is {VB,B} $\wedge$ Nd is {N}	$\Rightarrow$ Ls is {B}
R_{26}:	If Nd is {N}	$\Rightarrow$ Ibs is {N}
R_{28}:	If Nd is {S}	$\Rightarrow$ Ibs is {S}
R_{30}:	If Nd is {VS}	$\Rightarrow$ Ibs is {VS}

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

corresponding definition of VB (*Very Big*) 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 path tracing method, the set of rules is defined as shown in Table 2. These rules model the expert knowledge of an expert who has used a Path-Tracer rendering engine for 3 years.

Composing the Partial Results

From the partial results generated by the different agents after having rendering the work units, the master m_i is responsible for composing the final image. This process should not be directly made because slight differences between fragments can be distinguished when they are obtained from different rendering agents due to the random component of Monte Carlo–based methods (like path tracing). In Figure 7 the quality difference between adjacent work units with and without a blending

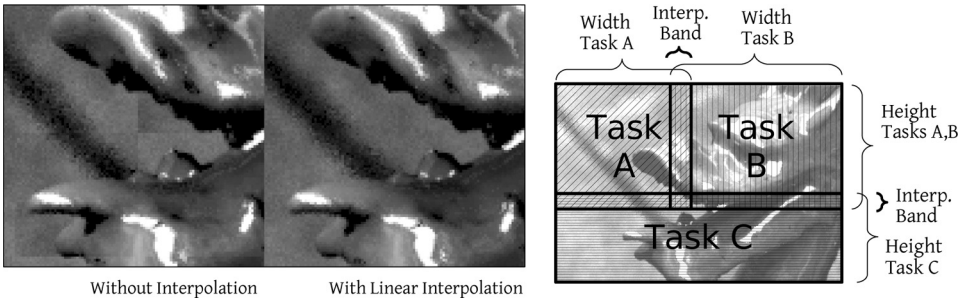

FIGURE 7 *Left*: The artifact appears with no interpolation between tasks. This problem is solved with linear interpolation. *Right*: Diagram of work unit division and interpolation band position.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

method can be appreciated. The image rendered with no interpolation shows important lack of continuity. For that reason it is necessary to smooth the joint between the work units that are neighbors by using a lineal interpolation mask. In this way, adjacent work units share a common area called Interpolation Band. A graphical example of the interpolation bands shared by three work units is also shown in Figure 7.

As previously described, the size of the Interpolation Band is an output parameter that is determined by the knowledge KBS_j of a rendering agent ra_{ij} depending on several input variables V_i of the work unit (Complexity and Neighbor Difference). Table 2 exposes the rules used by ra_{ij} to set the value of the interpolation band (see rules 22, 24, 26, 28, and 30). The value of this parameter should be larger if the quality difference between neighbor zones is important. To obtain better time results, the value of this parameter should not be large because the interpolation band implies that the rendering of a same area is performed by two or more agents. This is specially important if the zone is very complex because the cost of rendering this interpolation band is also very high. The amount of time wasted in the interpolation band is currently estimated between 2% and 5% of the global rendering time.

EXPERIMENTAL VALIDATION

Platform

The results reported in this section have been obtained with the implementation of MAgArRO, which we have made available for download from the official web page of the project,[1] under GPL Free Software License. To evaluate the system, 16 computers with the same hardware characteristics have been used. However, the architecture supports any

[1]http://www.esi.uclm.es/www/cglez/magarro

number of agents and the replication of the related services. The tests were run with different number of agents, partitioning characteristics, and optimization levels. It is important to remark that each one of the rendering agents ra_{ij} runs on a different computer since rendering essentially involves processing power. The computers are Pentium Intel Centrino 2 GHz, 1-GB RAM, and Debian GNU/Linux. The rendering method used in all cases was path tracing (Yafray 0.1.0 render engine), 8 oversampling levels, 8 recursion levels in global configuration of ray tracing, 1024 Light samples by default, and with Irradiance Cache activated. The dragon scene to be rendered contains more than 100.000 faces, 5 levels of recursion in mirror surfaces (Figure 10), and 6 levels in transparent surfaces (the glasses). Rendering the dragon scene with this configuration in a single computer took 121 minutes and 17 seconds.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Implementation Details

Because MAgArRO aims at optimizing the distributed rendering by means of a multiagent solution in which the rendering agents run on different computers (possibly over various operating systems and under different hardware platforms), we have used the middleware ZeroC ICE (Henning 2004) to develop the distributed system. Against other alternatives such as CORBA, we have chosen this modern communication framework due to its simplicity, efficiency, heterogeneity, and the set of advanced services provided.

As previously described, MAgArRO has been deployed on top of a general-purpose FIPA-compliant multiagent system. In this context, the basic management services (AMS, DF, and ACC) of the platform have been implemented in C++ to achieve a high efficiency when processing management functions, such as agent registration or communication issues. On the other hand, the agents are deployed within the system through agent factories, which have been implemented in three different programming languages: C++, Java, and Python. The goal of this approach is to provide the developer of the multiagent application with a wide variety of programming options when carrying out the development of domain-specific agents. In addition, the middleware ZeroC ICE allows agents implemented in different programming languages to communicate with one another.

The rendering agents have been implemented in Python and make use of the API of the Blender 3D suite to perform the rendering of the work units in which the input scene was divided. In addition, to specify the knowledge base of rendering agents we used the XML metalanguage, which allows us to easily and intuitively describe this knowledge. We also developed in Python the Mamdani's fuzzy inference method to apply the set of rules that constitutes the knowledge base of rendering agents.

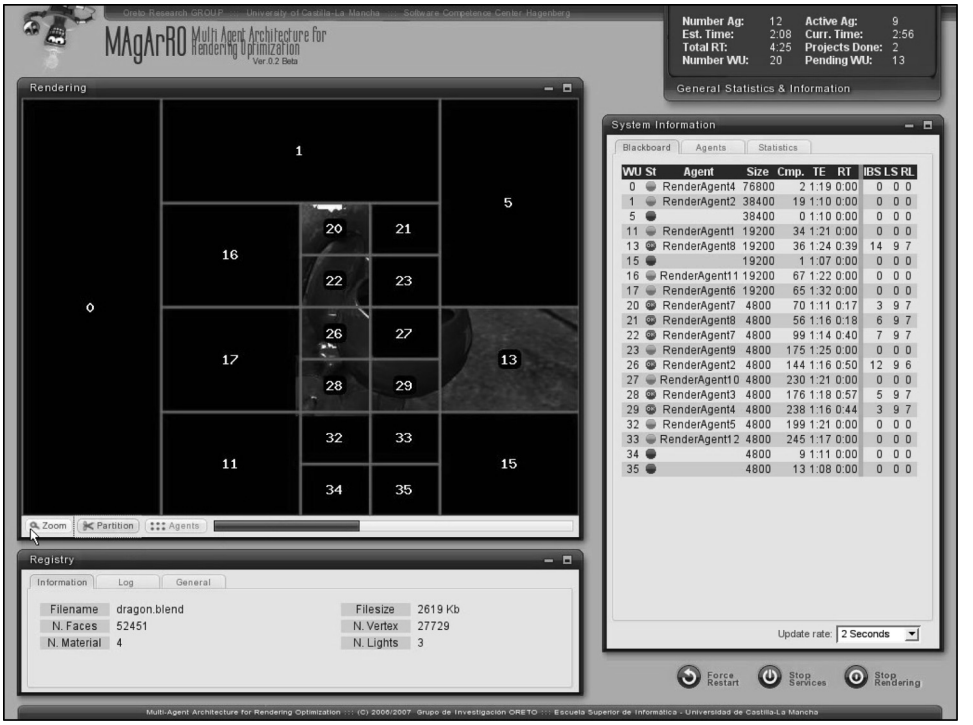

FIGURE 8 Screenshot of the MAgArRO web interface. *Left*: Current state of the rendering process. *Right*: Current state of the Blackboard.

Finally, and for the users to interact with MAgArRO, we developed a web system that allows to submit works from a web browser anywhere on the Internet. In this way, the user can observe in real-time the progress of rendering and the currently estimated remaining time. Figure 8 shows the visual aspect of this graphical user interface when MAgArRO is rendering a work previously submitted by the user.

TABLE 3 Times (mm:ss) with Different Partitioning Levels with the *Normal* Optimization Level in the Dragon Scene (Figure 10)

Agents	First level	Second level	Third 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
16	26:16	23:32	10:03

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Experimental Results

First, we study the importance of a good task division scheme. In Table 3 the time spent when using different partitioning levels in the dragon scene (Figure 10) is shown. These times have been obtained by using a normal optimization level. The top part of Figure 9 graphically represents these times. When using a simple partitioning of first level, we obtain a good rendering time with a few agents in comparison with the third level of partitioning. The time in the third level is longer because we are making more partitions in the areas with more complexity (in the glasses). This higher number of partitions involves interpolation more bands and, therefore, to repeat the rendering of some part of these areas in various work units. This situation implies that some of the more complex work is made more than one time and, for example, the rendering time with one agent in the third level is 105 minutes and in the

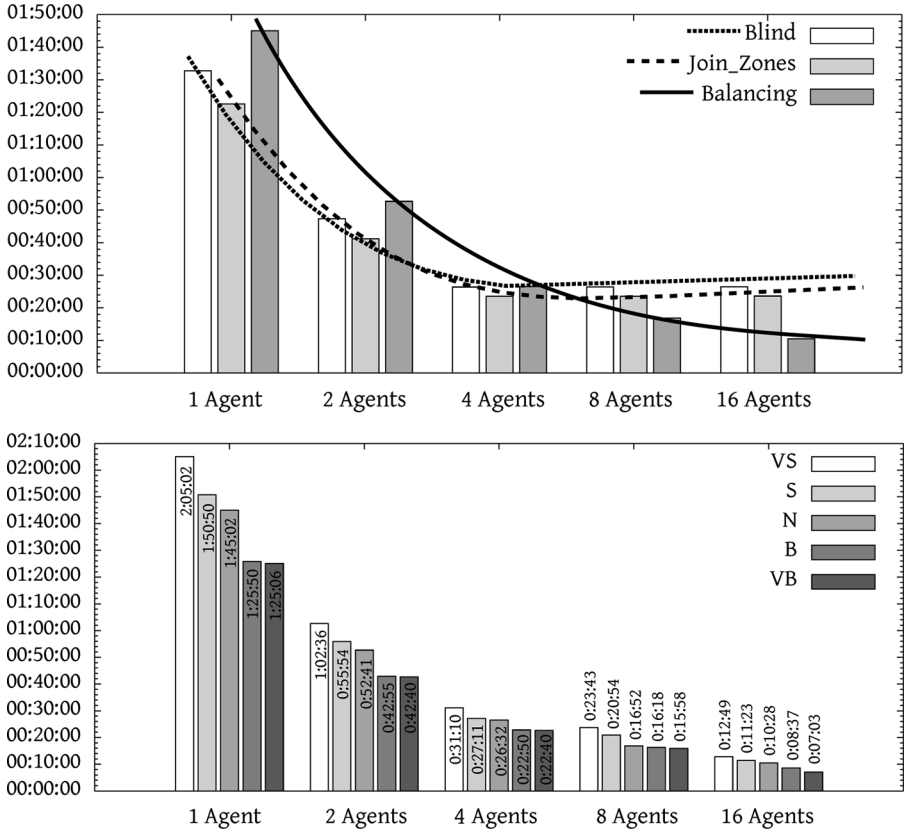

FIGURE 9 *Top*: Time (hh:mm:ss) required using different partitioning levels (normal optimization level) for the dragon scene (Figure 10). *Bottom*: Time (hh:mm:ss) required with different optimization levels (all with third level of partitioning) for the dragon scene (Figure 10).

TABLE 4 Times (mm:ss) with the Third Level of Partitioning (Balanced) in the Dragon Scene (Figure 10) with Different Number of Agents and Optimization levels

Agents	Very small	Small	Normal	Big	Very big
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
16	12:49	11:23	10:28	08:37	07:03

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

first level near 93 minutes. However, when the number of agents grows up, the performance of the system comes better because there are no too relevant differences of complexity among work units. In the partitioning of first and second level, there are complex tasks that slow down the finish of the whole rendering, and the increase in the number of agents does not produce relevant improvements (time required with four or eight agents

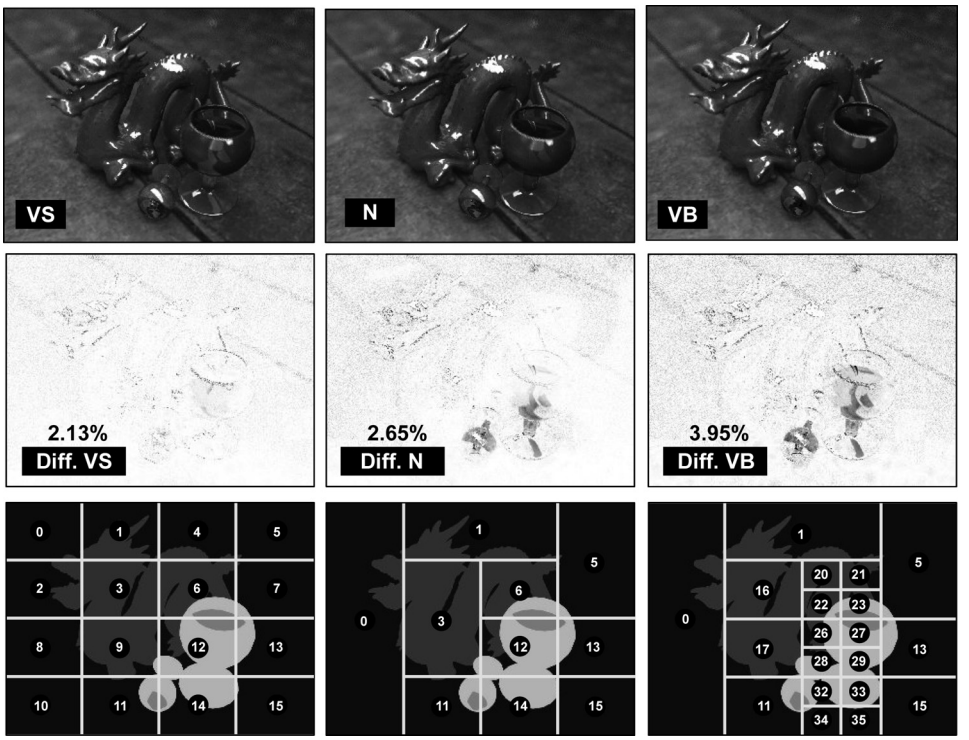

FIGURE 10 *First Row*: Results of the dragon scene rendering when using different optimization levels. *Second Row*: Quality differences between the results with and without optimization (the darker the bigger difference). *Third Row*: Three available partitioning levels when dividing the scene into work units.

are essentially the same). For this reason, the third partitioning level works better with a higher number of agents.

Table 4 represents the time spent in rendering the dragon scene when using different levels of optimizations, always using a third level of partitioning (see also Figure 9, bottom). If we simply use a small level of optimization, we obtain better results than if rendering is done with no optimization. The time required with very small optimization is greater than rendering the original scene because there is some time spent in the communication and composition of results, which, in this case, is greater than the time needed to render the scene only divided into one work unit with the original values.

The times are really good when only using four agents. As we can see for example in the normal optimization level, the time required to render the scene with four agents is only near 26 minutes in comparison with the 120 minutes of original rendering time. The rendering results with these configurations are shown in Figure 10.

To optimize the scene, different quality levels were taken. With some aggressive optimization levels (big or very big), there might be some lack of quality or loss of details. For example, in Figure 10 (first row, VB), the reflections on the glass are not so detailed like in Figure 9 (first row, VS). These errors might appear in different parts of the image (depending on the scene), so very high optimization levels may present this kind of problems. These artifacts might be more or less important and could be avoided by using more complex functions to develop the Importance Map and by choosing less aggressive optimization methods. In other projects these errors may be acceptable and the benefit may be worth the trouble. For instance, in the normal optimization level, the time needed to render the dragon scene is 26 minutes (120 minutes are necessary to render the original scene) and there is only a relative difference of the 2.65% of the pixels.

When studying the percentage of time spent in each step of the process (Figure 11), as it was predictable, the rendering stage uses most of the time (around 85%) and the estimation around the 10% (the time spent in other stages is negligible in comparison with these ones). The upload time is also very small because of the use of a LAN network. In other cases it may be significant but usually small in comparison with the rendering stage. As shown in Figure 11 (bottom), the time percentage needed by the stages when using a different number of agents is very similar. In this way, it is possible to use more advanced negotiation and cooperation mechanisms between agents because this might not significantly load the system in comparison with other times.

Other tests have been done by using the same hardware architecture and the same rendering method. Figure 12 shows the result of these

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

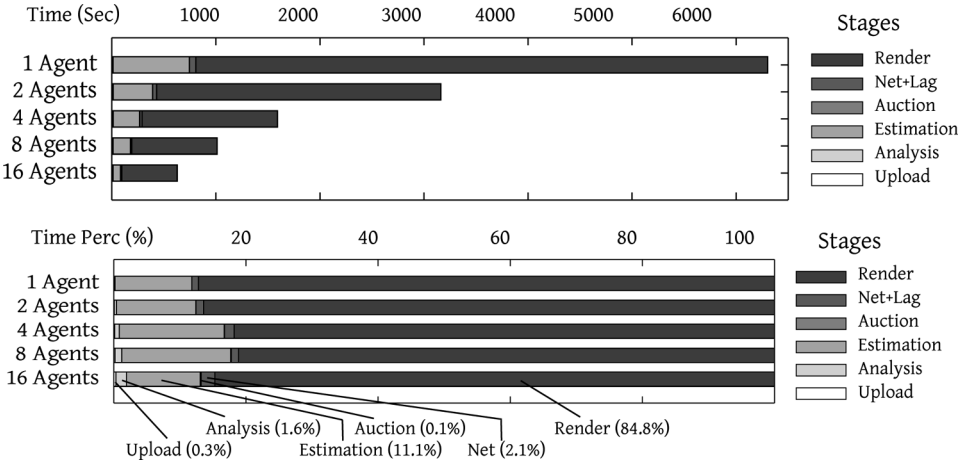

FIGURE 11 Resume of the dragon scene times. *Top*: Time spent in each stage with different number of agents. *Bottom*: Percentage of time spent in each stage.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

tests, whereas Table 5 resumes the time results. In all these scenes the global configuration was 1024 Light samples with Irradiance Cache, 8 oversampling levels, and 8 recursion level of path tracing. The room scene of Figure 12 contains more than 130,000 polygons, and it makes use of photon mapping (600,000 photons were shot from each light source), with four area lights in the scene. The robot scene of Figure 12 is an outdoor configuration similar to the dragon scene (Figure 10). It has more than 24,000 faces, and the light configuration (a general sky) is similar to the previous one, but in the dragon scene a high dynamic range image (HDRI map) has been used to achieve realistic lighting. Finally, the toilet scene of Figure 12 is another indoor scene, with only one light source (the light enters through the window), using 700,000 photons and with 12,000 faces. The complexity of this scene is due to the hard light configuration.

The result of the tests present in most cases achieves very good rendering times when only using four agents. Increasing the number of agents in some cases achieves a quasi-linear optimization in rendering times (like in the room or toilet scenes). This may be improved by choosing a smaller minimum size of work units by adopting a better importance map. Nevertheless, the results in any case are more than acceptable. With this optimization level, there is almost no perceptible difference between the original rendered scene (first row in Figure 12) and the result obtained with the normal optimization level (second row of images in Figure 12).

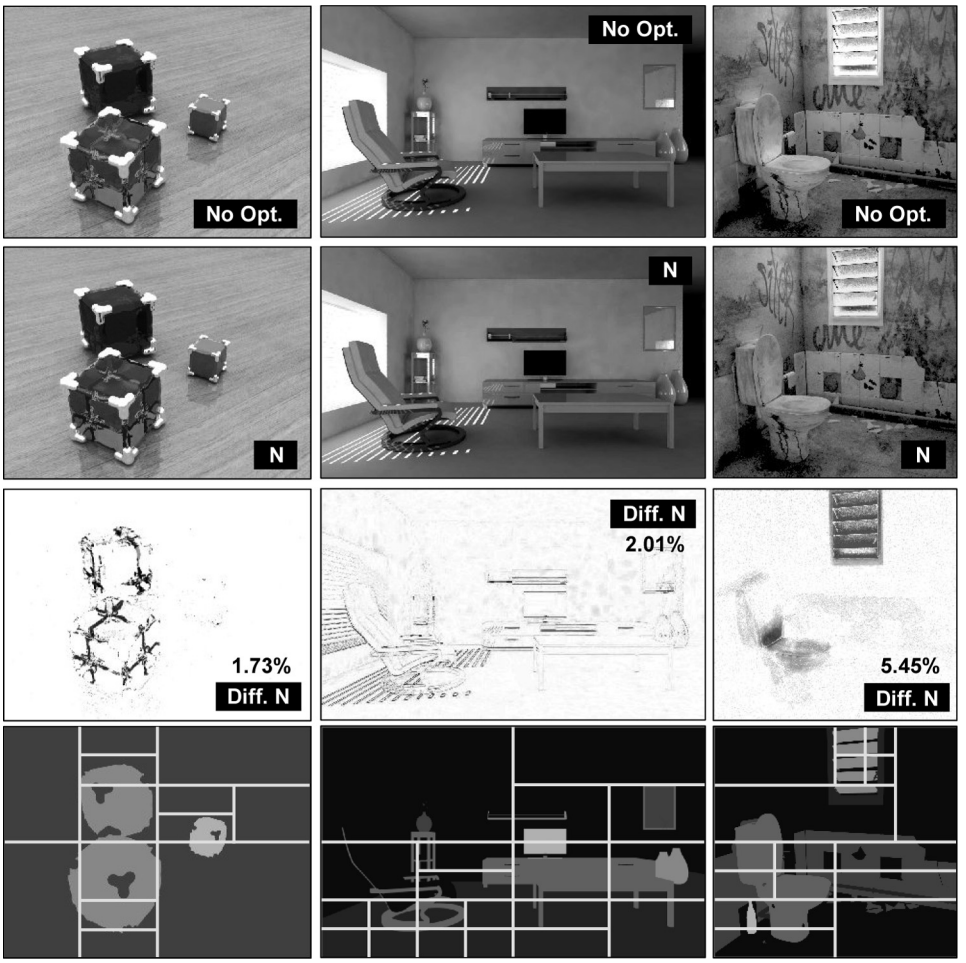

FIGURE 12 Other scene results. *First Row*: No optimization. *Second Row*: Normal optimization. *Third Row*: Quality differences between the results with and without optimization. *Fourth Row*: Partition for third level.

TABLE 5 Times Spent with the Third Level of Partitioning and with a Different Number of Agents. All Configurations, Except the First One with no Optimization, Use the *Normal* Optimization Level

Agents	Room scene	Robot scene	Toilet scene
1 *(No Opt.)*	26:59	32:40	29:49
1	25:32	27:11	25:47
2	13:27	13:52	13:19
4	6:58	7:04	7:53
8	4:19	3:58	4:02

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

CONCLUSIONS AND FUTURE WORK

The media industry is demanding for high-fidelity images for their 3D scenes. The production of 3D realistic images in computer games, film production, and TV advertisement was over 30,000 millions of dollars in 2005 (Roberts 2005), and it continues growing, becoming the most economically relevant activity in the leisure area. However, the computational requirements of full global illumination are such that it is practically impossible to achieve this kind of rendering in a reasonable time on a single computer.

To face this problem, the most widespread approach consists in using a rendering farm composed of hundreds or event thousands of computers to distribute the computational load of rendering. Other approaches involve the use of dedicated graphic processing units to accelerate the rendering. The main limitations of these approaches lie in achieving an effective load balancing and the need of redesigning the algorithm for each hardware architecture, respectively. On the other hand, the user tends to over-optimize the value of rendering engine parameters so that the final rendering time is considerably increased with no relevant improvement in the perceptual quality of the final image.

The approach presented here has been devised in response to these challenges. MAgArRO consists in a multiagent system that allows to distribute the work units in which an input scene is divided into rendering agents that run on different computers and use the knowledge bases needed to optimize rendering methods. MAgArRO delivers significantly better throughput (number of rendered work units per time unit) than classical (based on *cluster* or *render farm*) approaches because each independent work unit is rendered and optimized in parallel by different agents. Besides, there exist agents within the MAS responsible for performing an adequate division of the input scene (*Analyst Agent*) and for coordinating the rendering of a 3D model by a set of rendering agents (*Master Agent*). This logical division into responsibilities allows the agents to deal with the processes of dividing the tasks, coordinating the distributed rendering, and performing the rendering using knowledge bases in an autonomous way. To share information the architecture provides a *Blackboard* service where these agents can read and write data.

The Analyst performs a critical function in the whole process when dividing the input scene into work units by using Importance Maps and different division levels to balance the complexity of the work units. The use of the importance map assures an initial [*good*] time estimation that minimizes the latency of the latest task. In particular, as a result of the optimization obtained, MAgArRO achieves overall rendering times lower than the time required by one CPU divided by the number of agents. Within this context, the third partition level (see the third column of the

third row of Figure 10) presents the best results when dividing the scene into work units (see top of Figure 9).

The Master is responsible for coordinating the rendering agents through an auction mechanism, which allows them to bid for the most complex work units in first place to reduce the global rendering time. Besides, this mechanism awards to the agents that finish the task in a time T_R shorter or similar to the previously estimated T_E. This approach improves the task retrieval by rendering agents by taking into account their historical behavior, that is, the sequence of successes or failures previously done.

One of the main contributions of MAgArRO is the use of knowledge bases to optimize the rendering process by adjusting the relevant parameters involved in this stage of photorealistic image synthesis. Experimental results show that MAgArRO achieves excellent optimization performance by modeling expert knowledge or previous successful experiences through *fuzzy systems*. The use of fuzzy systems provides several advantages:

- It allows us to easily model expert knowledge. The definition of the sets of rules is based on XML files, which are easily adjustable with no changes in the source code.
- It provides us with a highly scalable system. Thus different types of agents can cooperate by using knowledge of different domains. The use of sets of fuzzy rules allows us to define various roles for the running agents in the system.
- It provides us with an understandable human representation. The management of uncertain and incomplete information is easily done with the use of fuzzy logic. This characteristic, typical of human reasoning, is difficult to achieve with other representation mechanisms.

MAgArRO is pioneering in the application of multiagent principles to 3D realistic rendering optimization. This fact opens several interesting research avenues. Specifically, we believe it is very promising to extend MAgArRO toward more sophisticated adaptation and machine learning techniques. The open technology used in the development of MAgArRO allows the agents to reside and run on different machines around the world.

There are several opened research lines related to MAgArRO:

- Choosing a **good configuration of the input parameters** for a given rendering engine is not an easy task. The quality of the synthesized image, as well as the CPU time spent in the process, depends to a great extent on the values of the input parameters of the rendering engine to

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

be used. One open research line should be oriented to the construction of a data mining-based rendering wizard, which automatically chooses a good initial configuration. To accomplish its objectives, the system would have a data repository that contains several matching models. In this way when the user inputs a model, the system tries to match the given model with one of the matching models in the data repository according to several similar criteria. Once one model has been matched, a data mining analysis is performed against its corresponding data set.

- More advanced importance functions could be used in this gray scale image generation, using perception-based rendering algorithms (based on visual attention processes) to construct the importance map (Cater et al. 2003; Sundstedt et al. 2005). The **importance map** can also be improved by using more information about the scene, such as geometric information about photon impacts, and direct and indirect illumination maps.
- We are also exploring other issues, such as the possibility of incorporating agent–agent real-time coordination schemes into MAgArRO, which are more flexible than auctioning, adding learning capabilities to the agents, and so on.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

REFERENCES

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

Anderson, D. P., E. Korpela, and R. Walton. 2005. High-performance task distribution for volunteer computing. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, 196–203. Washington, DC: IEEE Computer Society.

Bellifemine, F., G. Caire, D. Greenwood, and E. Corporation. 2007. *Developing Multi-Agent Systems with JADE*. New York: Wiley.

Buck, I., T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. 2004. Brook for GPUs: Stream computing on graphics hardware. In *Proceedings of SIGGRAPH 2004*, 777–786. Los Angeles, CA: ACM.

Castro, J., J. Castro-Schez, and J. Zurita. 1999. Learning maximal structure rules in fuzzy logic for knowledge acquisition in expert systems. *Fuzzy Sets and Systems* 101:331–342.

Castro-Schez, J., J. Castro, and J. Zurita. 2004. Fuzzy repertory table: a method for acquiring knowledge about input variables to machine learning algorithm. *IEEE Transactions on Fuzzy Systems* 12:123–139.

Cater, K., A. Chalmers, and G. Ward. 2003. Detail to attention: exploiting visual tasks for selective rendering. In *Proceedings of the 14th Eurographics workshop on Rendering*, 270–280. Aire-la-Ville, Switzerland: Eurographics Association.

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

Cook, R., T. Porter, and L. Carpenter. 1984. Distributed ray tracing. *ACM SIGGRAPH Computer Graphics* 18:137–145.

Fernandez-Sorribes, J., C. Gonzalez-Morcillo, and L. Jimenez-Linares. 2006. Grid architecture for distributed rendering. In *Proceedings of Ibero-American Symposium in Computer Graphics 2006 (SIACG'06)*, 141–148. Switzerland: Eurographics Association.

FIPA. n.d.a. Foundation for Intelligent Physical Agents FIPA. http://www.fipa.org (accessed March 29, 2010)

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010

FIPA. n.d.b. Foundation for Intelligent Physical Agents FIPA. http://www.fipa.org/specs/fipa00029/SC00029H.html (accessed March 29, 2010)

Foster, I., C. Kesselman, J. Nick, and S. Tuecke. 2002. The physiology of the grid: an open grid services architecture for distributed systems integration. Muncie, Ind.: Open Grid Forum.

Gillibrand, R., K. Debattista, and A. Chalmers. 2005. Cost prediction maps for global illumination. In *Proceedings of Theory and Practice of Computer Graphics'05*, 97–104. Eurographics Association. London: Local Events.

Gillibrand, R., P. Longhurst, K. Debattista, and A. Chalmers. 2006. Cost prediction for global illumination using a fast rasterised scene preview. In *Proceedings of the 4th International Conference on Virtual Reality, Computer Graphics*, Visualisation and Interaction in Africa, Afrigraph 2006, Cape Town, South Africa, January 25–27, 2006, 41–48.

Gooding, S., L. Arns, P. Smith, and J. Tillotson. 2006. Implementation of a distributed rendering environment for the TeraGrid. In *Proceedings of the IEEE Challenges of Large Applications in Distributed Environments*, 13–21. Paris: IEEE Computer Society Washington, DC.

Goral, C., K. Torrance, D. Greenberg, and B. Battaile. 1984. Modeling the interaction of light between diffuse surfaces. *ACM SIGGRAPH Computer Graphics* 18:213–222.

Günther, J., I. Wald, and P. Slusallek. 2004. Realtime caustics using distributed photon mapping. In *Proceedings of the Eurographics Symposium on Rendering*. Sweden: Eurographics Association.

Hachisuka, T. 2005. High-quality global illumination rendering using rasterization. *GPU Gems* 2:615–633.

Heath, T. 2008. The plush life. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Computer Animation Festival*, 86–87. New York: ACM.

Henning, M. 2004. A new approach to object-oriented middleware. *IEEE Internet Computing* 8:66–75.

Henning, M., and S. Vinoski. 1999. *Advanced CORBA Programming with C++*. Reading: Addison-Wesley.

Hurley, J. 2005. Ray tracing goes mainstream. *Intel Technology Journal: Compute Intensive, Highly Parallel Applications and Uses* 9:99–107.

Jensen, H. 2004. A practical guide to global illumination using ray tracing and photon mapping. In *International Conference on Computer Graphics and Interactive Techniques*. New York: ACM.

Jevans, D. 1989. Optimistic multi-processor ray tracing. In *CG International '89: New Advances in Computer Graphics*, 507–522. Germany: Springer-Verlag.

Kajiya, J. T. 1986. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, 143–150. New York: ACM.

Kato, T., and J. Saito. 2002. Kilauea: parallel global illumination renderer. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, 7–16. Aire-la-Ville, Switzerland: Eurographics Association.

Kerlow, I. V. 2004. *The Art of 3D Computer Animation and Effects*. New York: Wiley.

Kipfer, P., and P. Slusallek. 1999. Transparent distributed processing for rendering. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel Visualization and Graphics*, 39–46. Washington, DC: IEEE Computer Society.

Lafortune, E., Y. Willems, and H. Santo. 1993. Bi-directional PathTracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics' 93)*, 145–153.

Martino, J., and R. Köhling. 1992. Production rendering on a local area network. *Computers & Graphics* 16:317–324.

McEwan, A., S. Schneider, W. Ifill, and P. Welch. 2007. Domain specific transformations for hardware ray tracing. In *Communicating Process Architectures 2007: WoTUG-30: Proceedings of the 30th WoTUG Technical Meeting* 479. Ios Pr Inc. Amsterdam: IOS Press.

Patoli, Z., M. Gkion, A. Al-Barakati, W. Zhang, P. Newbury, and M. White. 2008. How to Build an Open Source Render Farm Based on Desktop Grid Computing. In *Wireless Networks Information Processing and Systems: First International Multi Topic Conference*, Imtic 2008 Jamshoro, Pakistan, April 11–12, 2008 Revised Papers, p. 268. Germany: Springer.

Pharr, M., and G. Humphreys. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann.

Rajagopalan, R., D. Goswami, and S. P. Mudur. 2005. Functionality distribution for parallel rendering. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Volume 1, 18–27. Washington, DC: IEEE Computer Society.

Rangel-Kuoppa, R., C. Aviles-Cruz, and D. Mould. 2003. Distributed 3D rendering system in a multiagent platform. In *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC '03)*, 168–175. Los Alamitos: IEEE Computer Society.

Ris, P., and D. Arques. 1994. Parallel ray tracing based upon a multilevel topological knowledge acquisition of the scene. In *Computer Graphics Forum*, Volume 3, 221–232. New York: Wiley

Roberts, K. 2005. *SISOMO: The Future on Screen*. New York: powerHouse Books.

Schlechtweg, S., T. Germer, and T. Strothotte. 2005. RenderBots—Multiagent systems for direct image generation. *Computer Graphics Forum*, Volume 24, 137–148. New York: Wiley.

Shirley, P., K. Sung, E. Brunvand, A. Davis, S. Parker, and S. Boulos. 2008. Fast ray tracing and the potential effects on graphics and gaming courses. *Computers & Graphics* 32:260–267.

Snell, Q., and J. Gustafson. 1997. Parallel hierarchical global illumination. In *High Performance Distributed Computing, 1997. Proceedings. The Sixth IEEE International Symposium on High Performance Distributed Computing*. 12–19. Washington, DC.

Stober, A., A. Schmitt, B. Neidecker, W. Muller, T. Maus, and W. Leister. 1988. Tools for efficient photo-realistic computer animation. In *Proceeding of Eurographics '88*, Volume 88. 31–41. Amsterdam, Elsevier.

Sundstedt, V., K. Debattista, P. Longhurst, A. Chalmers, and T. Troscianko. 2005. Visual attention for efficient high-fidelity graphics. In *Spring Conference on Computer Graphics: Proceedings of the 21st Spring Conference on Computer Graphics*, 12:169–175. New York: The Association for Computing Machinery.

Veach, E., and L. J. Guibas. 1997. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, 65–76. New York: ACM Press/Addison-Wesley.

Ward, G., F. Rubinstein, and R. Clear. 1988. A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, 85–92. New York: ACM.

Weiler, M., and T. Ertl. 2001. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In *Proceedings of the Conference on Visualization'01*, 199–206. Washington, DC: IEEE Computer Society.

Weiss, G. 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, Mass.: MIT Press.

Whitted, T., and N. J. Holmdel. 1980. An improved illumination model for shaded display. *Communications* 23:343–349.

Woop, S., J. Schmittler, and P. Slusallek. 2005. RPU: a programmable ray processing unit for realtime ray tracing. In *International Conference on Computer Graphics and Interactive Techniques*, 434–444. New York: ACM Press.

Zadeh, L. 1999. Fuzzy logic = computing with words. *Computing with Words in Information/Intelligent Systems* 1:3–23.

Zhukov, S., A. Iones, and G. Kronin. 1998. An ambient light illumination model. In *Proceedings of Eurographics Rendering Workshop98*, 45–55. NewYork: Springer-Wien.

Downloaded By: [Morcillo, Carlos Gonzalez] At: 07:09 13 April 2010