Ibero-American Symposium on Computer Graphics - SIACG (2006)
P. Brunet, N. Correia, and G. Baranoski (Editors)

Grid Architecture for Distributed Rendering

J. A. Fernández-Sorribes, C. González-Morcillo, L. Jiménez-Linares

Escuela Superior de Informática, Ciudad Real
University of Castilla-La Mancha, Spain

Abstract

Rendering is the process by means of which a raster 2D image can be obtained from the definition of a 3D scene. This process is computationally intensive and requires a lot of time to be done when the source scene has certain complexity or when high-quality realistic images are required.
YAFRID (Yet Another Free Render grID) is a system that takes advantage of the characteristics of computational grids by distributing the rendering of a scene among a large number of heterogeneous computers connected to the Internet. With that kind of systems, the time a scene takes to be rendered is drastically decreased because the parts in which the complete work has been divided (called workunits) can be processed in parallel and finally joined using an interpolation function to obtain the image result.
As we will discuss on the final section of this article, the selection of the workunit size is a key step in this class of work division. With this approach, local optimizations in each workunit could be made to obtain a better rendering time per frame.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Distributed/network graphics I.3.3 [Computer Graphics]: Parallel processing I.3.7 [Computer Graphics]: Raytracing

1. Introduction

The process of creating a 3D animation comprises several phases such as character design and modeling, setting textures and materials, construction of the bones and subsequent animation, lighting, and, finally, rendering.

The latter consists basically in generating a 2D image from the abstract description of the geometry of a scene plus the definition of lights, cameras, and materials. This usually is the most computationally intensive phase of the whole process and, as a result, it takes a long time to be done. That situation is even worse when the scene to be rendered is complex or when high-quality realistic images are required. Rendering is often considered to be a bottleneck in that kind of projects and even obtaining just one image could need a lot of rendering time.

In spite of the fact that the huge time dedicated to render a scene represents an important problem, rendering is also a highly parallelizable activity because each frame of the whole animation can be calculated independently of the others. In the case of static scenes, an image could be divided into fragments to be processed separately.

In order to solve such a key problem in animations, and taking into account the advantage mentioned, several approaches has been developed. Today, the most popular one is the well-known **render farm**. A render farm is a computer cluster owned by an organization where each frame of an animation is independently calculated by a single processor.

Over the last few years, 3D animation companies have been discovering a new source of benefits in the Internet. Besides using its render farms in its own productions, some of this companies offer render services via Internet. An user of these render services can make use of the dedicated cluster that the company owns. The main difference between them and the project which this document is about (apart from the frame-level division and interpolation schemes) is that Yafrid allows anyone to be part of the grid by sharing CPU cycles via the Internet.

This technique and other similar ones work by dividing a complex task (rendering a complete animation or scene) into a set of smaller tasks (rendering a frame or a fragment of a frame). As a result, the time dedicated to render a whole scene is highly decreased because each smaller task is done in parallel with the other ones.

As mentioned, it is possible to take advantage of a parallel processing not only dividing an animation into independent frames but also dividing a static frame into smaller pieces. Each one of these units can be sent to a computer to be done independently and in a parallel way with the other ones. Once all the units has been finished, all of them has to be composite in order to obtain the whole image. Therefore, there are two types of distributed rendering systems according to its task granularity: *fine-grained* and *coarse-grained* (or frame-level).

The system architecture this document is about has taken both approaches as it supports both types of granularity.

2. Grid Computing

2.1. Definitions

There are several definitions of what a grid system is but all of them are based on the same basic principles.

It was in 1998 when the term grid computing came up and Carl Kesselman and Ian Foster gave the following definition [FK98]:

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

The evolution of this definition led to a checklist given by Ian Foster [Fos02]. According to this list a Grid is a system that:

1. coordinates resources that are not subject to centralized control ... *(A Grid integrates and coordinates resources and users that live within different control domains, different administrative units of the same organization or different companies. This model is often described with the term virtual organization which is profoundly analyzed in [FKT02])*
2. ...using standard, open, general-purpose protocols and interfaces ... *(A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. It is important that these protocols and interfaces be standard and open.)*
3. ...to deliver nontrivial qualities of service. *(A Grid allow its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.)*

Another definition due to Rajkumar Buyya appears in a recent article [BS05] and defines a grid as *a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.*

2.2. Peer-to-peer networks, clusters and grids

There are some terms such as distributed systems, peer-to-peer networks, and clusters which are related with the concept of grid computing and are often mistaken. All these terms have obvious similarities and also several differences [LSSH03]. For that reason, a definition of what a grid is and what is not is essential.

Like peer-to-peer, grid computing allows users to share files, and additionally grid allows many-to-many sharing for all types of resources.

Clusters, distributed computing, and grids have in common that all these approaches make a large quantity of resources appear as one single computational resource. However, unlike clusters and distributed computing, which need physical proximity and operating homogeneity (and are often owned by a single organization), grids can be geographically distributed and heterogeneous.

2.3. Grid classification

There are several classifications of grids according to different characteristics. One of them is given by IBM [Bro02] and classifies this kind of systems according to their functionality. This classification establishes that there are two primary grid types, *computational* and *resource*, and one common hybrid type, the *application grid*.

- **Computational grid.** Those systems are focused to solve computationally intensive problems. Computational grids make use of the CPU and usually memory components of the grid to increase the overall computing power.
- **Resource grid.** The aim of this grids is to manage and distribute vast quantities of data. A resource grid is used to store information across a range of computers, either to improve performance in retrieval, or to expand the potential storage space.
- **Application grid.** Rather than providing access to a specific resource, application grids use systems from both the computational and resource grids to support their functionality.

Another common classification [LMV*04] takes into account the shared resources that the grid contains. Resources to be shared can be computation resources (CPU cycles and storage), information resources (data bases and application data), displays, instruments, etc.

Reinefeld and Schintke [RS02] give another classification and distinguish three categories of grid systems: the Information Grid, the Resource Grid, and the Service Grid.

- The **Information Grid** is equivalent with today's world wide web that delivers information on any kind of topic to any place in the world. Information can be retrieved mediated by many different kinds of networking infrastructure, personal computers, advanced mobile phones, etc. Also file sharing services are part of the information grid.
- The **Resource Grid** provides mechanisms for the coordinated use of resources like computers, data archives, application services, and special devices. These grids give access to resources without bothering the user with the name, location and other attributes of the resources used. In contrast to the data supplied by the Information Grid, the facilities of the Resource Grid are supplied to authorized users only.
- The **Service Grid** delivers services and applications independent of their location, implementation, hardware platform, etc. The services are built on the concrete resources available in the Resource Grid. While the Resource Grid gives access to concrete resources, the Service Grid provides abstract, location-independent services.

3. Other related systems

As it has been mentioned, the problem that represents rendering an scene, due to its characteristics, is suitable to be solved using distributed approaches. To deal with that kind of problems, clusters of computers are commonly used by companies dedicated to graphics. Those systems require that all computers belong to the same organization with a small level of heterogeneity. Two examples of this approach are the distributed rendering environment at Purdue University [MAB05] and the solution of DeMarle et al. [DPH*03], using a cluster of PCs.

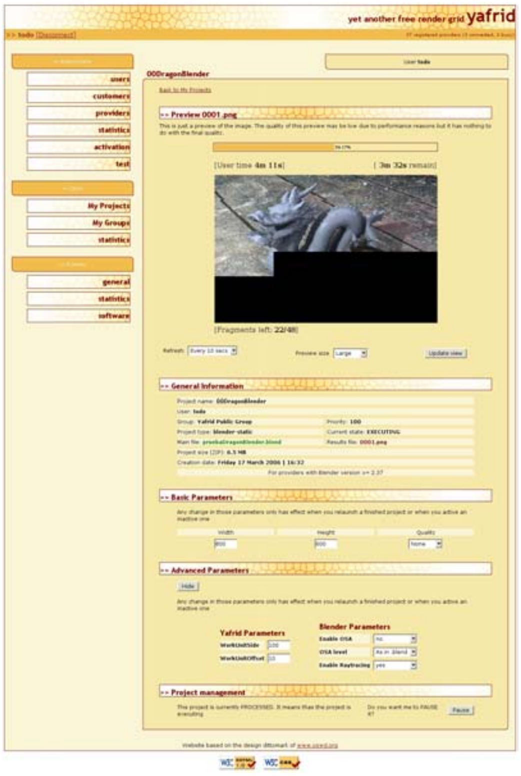
On the other hand, grid computing is an emergent field and, nowadays, that kind of computing approach is taken to solve an increasing number of computationally intensive problems through the Internet.

One of the most famous examples of these systems is SETI@home, pioneering scientific project dedicated to the Search for Extraterrestrial Intelligence (SETI). This experiment harnesses the power of hundreds of thousands of Internet-connected computers to analyze the data from the Arecibo Radio Observatory in Puerto Rico.

After that one, several applications have found in grid approaches a solution for complex operations performance. Another grid based solution to visualize tomographic data is RAVE [GAW04], that is mainly oriented to obtain an interactive representation of large volumes of data.

The main contribution of the system that this article presents consists in changing the classic approach to the problem of rendering, clusters, by a grid computing approach where everyone can be a provider of CPU cycles, in a non-dedicated, heterogeneous global architecture. The theoretic power of this approach is huge because any computer (it is a multiplatform system in terms of both hardware and software) with an Internet connection can be part of the grid, in the typical distribution of peer-to-peer applications.

Another difference that provides an advantage over other related works is that Yafrid supports both kinds of granularity. It permits to divide the rendering of an animation into frames or the rendering of a frame into fragments using the same key concept, the workunit.



00DragonBlender

-- Preview 0001.png

[User time 4m 11s] [3m 32s remain]

[Fragments left: 22/48]

Refresh: Every 10 secs Preview size: Large

-- General Information

Project name: 00DragonBlender
User: tedo
Group: Yafrid Public Group Priority: 100
Project type: blender-static Current state: EXECUTING
Results file: 0001.png
Project size (ZIP): 6.5 MB
Creation date: Friday 17 March 2006 | 16:32

-- Basic Parameters

-- Advanced Parameters

Yafrid Parameters
WorkUnitSide
WorkUnitOffset

Blender Parameters
Enable OSA
OSA level
Enable Raytracing yes

-- Project management

Figure 1: *Yafrid Web User Interface.*

4. Objectives

One of the aims of the project is to create an extensible architecture which supports as many render engines as possible (in its current stage, the system has support for the Open Source render engines Yafray and Blender). The system also supports fine-grained and coarse-grained task division to implement methods for optimizing the rendering time using a Grid network.

The system is aimed to spread the computing requirements of rendering a scene across a large number of computers, as many as possible. In order to achieve this goal, the grid has been developed to be multiplatform in terms of both hardware and software (operating system). These feature and the operation via the Internet, makes the number of potential service providers almost unlimited.

As a result, Yafrid is basically a system which take advantage of the characteristics of the computational grids by distributing the rendering of a scene through the Internet. Besides this basic functionality, there are other activities that are not the primary ones but are also important. These tasks (see User Interface in Figure 1) are activities related with the management of the workunits and with controlling the grid such as the management of client and provider priorities, groups, etc.

5. General Architecture

The type of grid which is the most suitable one for distributing the rendering of a scene among several computers is the computational grid. It is deduced from the properties of rendering considering it as an intensive process in terms of use of CPU.

The components of a grid of this type are basically the following ones:

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

5.1. Yafrid server

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

Yafrid server is developed over an architecture in which, in general, four layers can be distinguished (Figure 2). This design is slightly based on the architecture that appears in [FKT02]. Those layers are *'Resource Layer'*, *'Service Layer'*, *'Yafrid Server'* and *'User Layer'* from lowest to highest level of abstraction. It is in the third one of these layers where the actual Yafrid server resides. The other levels are considered to be supporting layers but are also indispensable for the implantation of the system in an actual environment.

- **Resource Layer**
  This layer has the lowest abstraction level and it is the most related with the operating system issues. Resource layer has the following components:
  - Database system. It is in this database where the necessary tables for the correct operation of the system are maintained. Some of these tables are used to obtain statistics about the system performance while other ones store the data associated to users, groups, projects, etc. This database is accessed from the levels above by means of a database server. The current implementation uses MySQL.
  - Filesystem. Sometimes, it is necessary to access directly to the filesystem from the layers above. Basically, the system distinguishes two types of directories. There are some directories which are used to store the workunits of the launched projects that will be accessed via SFTP by providers. Those directories compose the workunits POOL. The other category of directories is composed by those ones that contains the information about the users and their projects.
  - Network system. The module dedicated to the communications that belongs to the main layer hides the utilization of the network resources of the computer by using a middleware (the current implementation uses ICE).
- **Service Layer**
  Basically, this layer contains the different servers that allow modules from the layers above to access the resources that belongs to the layer under this one. There are the following servers in this level:
  - HTTP Server. Yafrid-WEB module is established over this server. As Yafrid-WEB has been developed using dynamic web pages written in a web-oriented scripting language (the current implementation has been done using PHP), the web server has to have support for this language. It is also necessary to have support for composition of graphics and for accessing to the database.
  - Database server. This server is used by the different modules of Yafrid to access the indispensable data for the system operation.
  - SFTP server. Accessed by the service providers to obtain the necessary files to carry out the rendering of the workunits. Once the rendering has finished, the SFTP server will be used to send to the Yafrid server the resultant image.
- **Yafrid Layer** This is the main layer of the server and it is composed by two different modules which work independently one of the other. These modules are Yafrid-WEB and Yafrid-CORE.
  - **Yafrid-WEB.**
    It is the interactive module of the server and it has been developed as a set of dynamic web pages written using HTML and a wed-oriented scripting language.
    In terms of access to the system, three user roles has

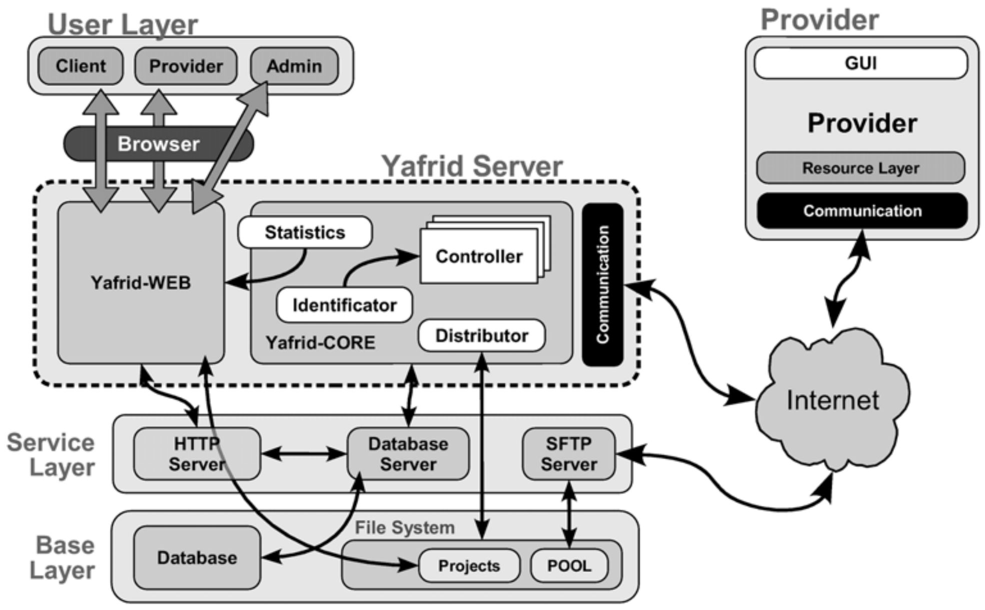


Figure 2: *Yafrid General Architecture.*

been defined which determine the access privileges that the user has. Those user roles are:

○ **Client.** Having this role allows an user to submit works to the grid. A client is also able to create and manage render groups which are groups to which clients and providers can subscribe. When a project is created, it can be created as belonging to a group. In this case, only providers belonging to the same group can take part in the rendering of the project. Moreover, some statistics are generated with the information of the projects.

○ **Administrator.** This user is necessary to operate the whole system and has complete privileges which implies:

  ◇ Access to the information about all the users of the system (providers and clients) and about the existing render groups.
  ◇ Creation of sets of tests (a special type of composed projects which are made up of single projects with slight differences). Those tests are tools by means of which conclusions about the system performance can be reached.

○ **Provider.** The provider is a user that has installed the necessary software to allow the grid to sent works to be done. Providers can access their information and some use statistics.

The mentioned render groups have a special importance in such a render grid. With this simple control, projects can be created within a group in order to have a set of providers working for them and not for other projects.

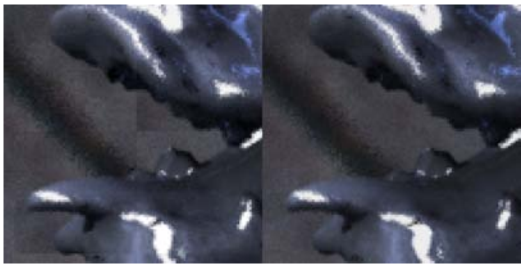


Figure 3: *(Left) Artifacts without interpolation between workunits.(Right) Using lineal interpolation*

– **Yafrid-CORE.**
This is the non-interactive part of the server. This module has been mainly developed using Python but there are also some scripts written in Bourne shell for management tasks. It is composed by three submodules:

  ○ **Distributor.**
  This is the active part of the server. It implements the main algorithm that is aimed to do the following indispensable tasks:

    ◇ **Generating the workunits**. It basically consists in launching the active projects that exist in the system by generating the necessary workunits according to the parameters that the user has introduced for the launching.

◇ **Assigning** the generated **workunits to providers**. This is the main task of the scheduling process. The algorithm must take into account issues such as the software that will be necessary to render the scene, the software installed in the providers and its specific version or the group which the provider and the project belong to. With all this information, the distributor has to decide which workunit goes to which provider.

◇ **Composing results**. With the results generated by the different providers, the distributor has to compose the final image. Each 10 seconds, the core could generate a preview with the available fragments (as shown in Figure 1). This process is not trivial because slight differences between fragments can be distinguished when obtained from distinct computers due to the random component of Monte Carlo based methods (like Pathtracing). For that reason, it is necessary to smooth the joint between fragments which are neighbors using a lineal interpolation mask. We define a zone in the workunit that is used to combine with other workunits in the server. In Figure 3 on the left, we can see problems when joining the workunits if we don't use a blending method.

◇ **Finishing projects** and deletion. All the partial results are deleted from the system, the files in the POOL, etc.

◇ **Timeout control**. When timeout is over, it is check if the workunits sent to the providers are still being processed or not. If any of the workunits has got lost it has to be activated again to be sent to another provider.

○ **Identificator.**
This is the passive part of Yafrid-CORE whose mission consists in waiting for the communications from the providers. This is the module that is responsible for the necessary protocol between the server and the providers to guarantee the correct operation of the system. It maintains all the communications with the providers except the one where a workunit is sent to a provider, which is done by the distributor.
The first time a provider try to connect to the Yafrid server the Identificator generates an object, the provider controller, and returns a proxy to this object. Each provider has its own controller. The subsequent interactions between the provider and the Yafrid server will be made by means of this controller. Some of these interactions are Identification, Subscription, Activation, Reporting the finalization of the rendering and Disconnection.

○ **Statistics.**
The statistics that the system stores have mainly two sources. There are some sections in Yafrid-WEB that show statistics about the operation of the system which are generated from the database. On the other hand, there are also a submodule that belongs to Yafrid-CORE that registers statistics which depend on the time like, for example, the users that have been connected for the last half an hour.

- **User Layer**
This layer is not part of the system in a strict sense. Basically consists of the different types of users that can access the system. There are three roles in Yafrid:

  – Yafrid Provider.
  – Yafrid Client.
  – Yafrid Administrator.

  These users can access all the information related to the system which is offered by the module Yafrid-WEB by means of any browser (Figure 1). The characteristics of each one of these roles and its interaction with the system will be explained in next sections.

5.2. Provider

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

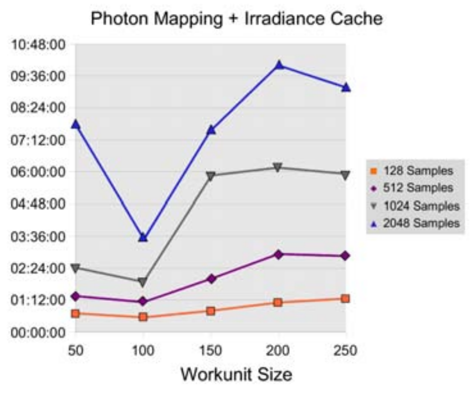


Figure 4: *Results with Photon Mapping.*

6. Results

In order to test the behavior of the system, 16 computers with the same characteristics have been connected to the grid. These providers (Pentium IV 2.80 GHz, 256Mb RAM) were connected to the grid during the execution of all tests. Each test was composed by a set of individual projects that differs in the size of the workunit. Each test had its own quality parameters (like the number of light samples or number of rays per pixel) and rendering method.

The image to be constructed using Pathtracing with Photon Mapping [Jen01] and Irradiance Cache (Figure 6) and classical Raytracing [Whi80] with an Ambient Occlusion implementation (Figure 7) was a bit complex. The scene contained more than 100.000 faces, 4 levels of raytracing recursion in mirror surfaces (the dragon) and 6 levels in transparent surfaces (the glass). Finally, 200.000 photons were shooted in order to construct the Photon Map structure.

Before studying the results obtained with the grid of 16 computers, we will present the rendering time of each test using only one provider (rendering in local machine):

Method	Quality	Time (hh:mm:ss)
Pathtracing	128 Samples	02:56:56
Pathtracing	512 Samples	07:35:28
Pathtracing	1024 Samples	14:23:07
Pathtracing	2048 Samples	23:16:12
Classic. RT.	1 Ray/Pixel	00:17:50
Classic. RT.	8 Rays/Pixel	00:20:39
Classic. RT.	16 Rays/Pixel	00:26:47

As we can see in Figure 4, the rendering time in the best case is more than eight times better using the grid, and only twice in the worst case. The results could be better if the render engine stored the irradiance cache to share it between providers (but this is not our case).

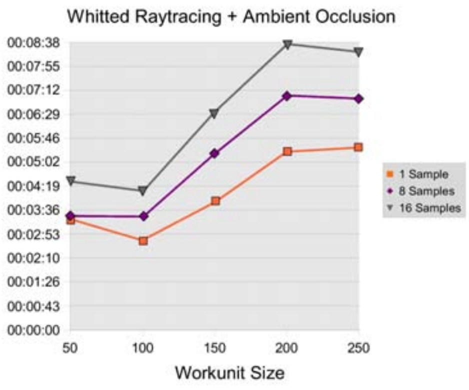


Figure 5: *Results with Classical Raytracing.*

Analogous to previous analysis, using Whitted raytracing algorithm (Figure 5), the rendering time in the best case is seven times smaller using the grid, and only three times better in the worst case.

With these results it is clear the importance of choosing an appropriate workunit size (it's also critical to achieve a good performance in a single-frame division method).

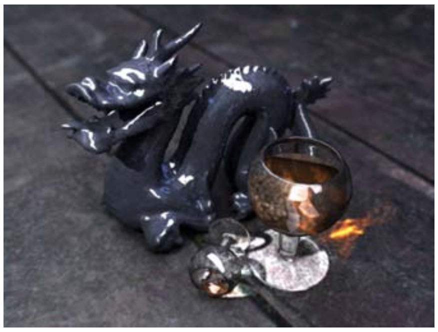


Figure 6: *Image generated with Pathtracing (512 Samples).*

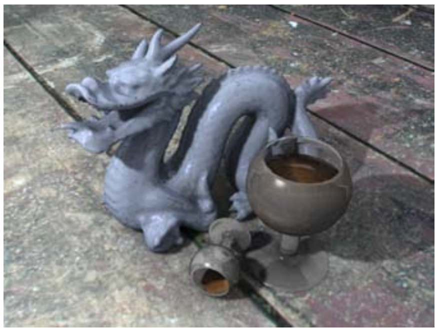


Figure 7: *Image generated with Classical Raytracing (1 Sample).*

7. Conclusions and future works

According to the results obtained that have been examined before, the system has clearly a better performace than a single computer system even in the worst selection of the workunit size.

Yafrid delivers significantly better **throughput** (number of rendered entities, frames or fragments of frame, per time unit) than a single computer because each independent unit is generated in parallel by different providers. This throughput depends on the workunit size.

On the other hand, the **latency** of each of the fragments (time they take to be done) increases because there are some issues that don't have to be taken into account in a single computer. In a distributed system like this one, besides the time dedicated to render the scene, also the *transfer time* (the time for the completion of the data transmission between server and providers) is important. Transfer time depends on the amount of data to transfer and the speed of the network connection.

Generally, the more fragments a scene is divided into, the more parallelism is achieved, and the less time is needed to render the whole scene. It is not always accurate in some cases. If a scene is divided into such small pieces that the time that rendering takes is inconsiderable, it would be possible that the time spent transferring and processing is bigger than the time dedicated to render. In this case, the results obtained by a distributed system are worse than the ones that a single computer can supply. In other cases, selecting a bad workunit size can degrade the throughput of the grid.

Yafrid is the first system of this characteristics, having some important advantages:

- There is **no cluster**. What it means is that the providers can be heterogeneous (in terms of software and hardware) and geographically distributed (like peer-to-peer applications).
- With the fine-grained approach, we can make **local optimizations** in each frame. In future works, we could study the scene and make an intelligent partitioning of the workunits, rendering each one with different quality settings. With this approach and using N machines in the grid, if X is the time taken by one machine to render the whole scene, we could obtain a rendering time smaller than X/N.
- One of the main advantages of this distributed approach is its **scalability**. The performance perceived by the user depends on the number of subscribed providers and the characteristics of them in a global way. The more and better providers Yafrid has, the better results will be obtained.
- Yafrid is Free Software. This system could be used in any research work or working in production projects.

7.1. Future works

An enhancement that would improve the performance of the system would consist in making an analysis of the scene before scheduling. In this previous step it would be analyzed how long each part of the scene takes to be done. According to the information obtained, the system would be able to use variable granularity when dividing a frame into fragments to increase efficiency.The zones of the scene which take more time to be done will be divided into smaller pieces in order to decrease the output time by balancing the time each fragment takes.

Each workunit could be calculated using different quality settings. In this way we can obtain better global rendering time (the sum of rendering time of each machine) than constructing the image with one processor.

8. Acknowledgement

This work has been funded by the Spanish ministry of Science and Technology under Research Project ”DIMO-CLUST” TIC2003-08807-C02-02.

References

[Bro02] BROWN M. C.: Buil a Grid App with Python. *ibm.com/DeveloperWorks* (2002).

[BS05] BUYYA R., SRIKUMAR V.: A Gentle Introduction to Grid Computing and Tecnologies. *CSI Communications 29*, 1 (July 2005), 9–19.

[DPH*03] DEMARLE D., PARKER S., HARTNER M., GRIBBLE C., HANSEN C.: Distributed interactive ray tracing for large volume visualization. *Proc. IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), 87–94.

[FK98] FOSTER I., KESSELMAN C.: *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, Elsevier, 1998.

[FKT02] FOSTER I., KESSELMAN C., TUECKE S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications 15*, 3 (2002).

[Fos02] FOSTER I.: What is the Grid? A Three Point Checklist. Web pages `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`.

[GAW04] GRIMSTEAD I. J., AVIS N. J., WALKER D. W.: Automatic distribution of rendering workloads in a grid enabled collaborative visualization environment. *ACM/IEEE SC 2004 Conference (SC'04)* (2004), 32–38.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis using Photon Mapping*. A.K.Peters, 2001.

[LMV*04] LEISTER W., MAZAHER S., VESTGÅRDEN J. I., JOHANSEN B., NORDLUND B.: Grid and related technologies. *Norwegian Computer Center Report* (June 2004).

[LSSH03] LEDLIE J., SHNEIDMAN J., SELTZER M., HUTH J.: Scooped Again. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)* (Berkeley, CA, USA, February 2003).

[MAB05] MADHAVAN K. P. C., ARNS L. L., BERTOLINE G. R.: A distributed rendering environment for teaching animation and scientific visualization. *IEEE Computer Graphics and Applications* (2005), 32–38.

[RS02] REINEFELD A., SCHINTKE F.: Concepts and Technologies for a Worldwide Grid Infrastructure. *Lecture Notes in Computer Science 2400* (2002), 62–71.

[Whi80] WHITTED T.: An improved illumination model for shaded display. *Communications of the ACM 33*, 6 (June 1980), 343–349.