On Harnessing Desktop Grids for Semi-Real Time 3D Rendering: A Case Study on POV-Ray

A. M. Riad[1], A. E. Hassan[2], Q. F. Hassan[1]
[1]Department of Information Systems, Faculty of Computers and Information Systems, Mansoura University, Mansoura, Egypt
[2]Department of Electrical Engineering, Faculty of Engineering, Mansoura University, Mansoura, Egypt

ABSTRACT: As is known, 3D rendering puts a huge burden both on computers and artists/programmers. This task is usually slow and requires the use of computational clusters as well as the installation of expensive resources such as multi-processors/cores and optimized graphics processing units to the traditional computers to lessen this burden. Parallelizing the rendering task of complex frames and scenes can cut the total rendering time dramatically. Grid computing is one of the parallel computing paradigms that allow implementers to combine the power of networked computers into a virtual supercomputer. Desktop grids are a form of grid computing that allows enterprises to build grids out of the workstations they already have. This feature could enable enterprises to better utilize their computing resources in order to solve computationally-intensive problems in a quick and cost-effective manner. The purpose of this paper is to present how enterprises can build desktop grids in a Windows environment in order to enable semi-real time rendering for 3D models defined with POV-Ray. Algorithms, tools and technical details are presented for easy and efficient implementations.
KEYWORDS: Grid Computing, Computer Graphics, Desktop Grids, Ray Tracing and POV-Ray,.NET Framework, Alchemi, Semi-Real Time 3D Rendering.

Introduction

Computer graphics is one of the most interesting and challenging branches in the field of computer science. Both researchers and professionals work toward

making the best use of computing resources to produce high-quality outputs that could be directly and indirectly made use of. Visual effects and movies are examples of direct uses of computer graphics, whereas 3D simulations and representations of protein folding during the design of new drugs are just a few examples of the indirect uses of computer graphics.

As is known, 3D rendering is a slow and compute-intensive process that usually comprises a large number of complex calculations. These calculations aim at defining the features to be included in the final output such as lighting, shading, fogging, shadows, texture mapping, reflection, refraction, diffraction, illumination, transparency, bump mapping and many other features.

This limitation has pushed companies that work in the field of 3D animation to look for solutions that would help them slash the rendering time. For instance, some companies had to buy really expensive hardware components like supercomputers and accelerated graphics processing units (GPUs) to save rendering time. Other companies outsource rendering process to commercial rendering services available online like RenderCore (http://www.rendercore.com/), RenderFlow (http://www.renderflow.com/), RebusFarm (http://www.rebusfarm.net/index.php/en/) and ResPower (http://www.respower.com/index.php). Such services deploy powerful farms that are composed of hundreds of computers solely dedicated to the rendering process. Companies that cannot afford any of these solutions usually ask their artists to render models after working hours (in forms of night batches) so that they can focus on designing these models during working hours. They are also forced, in many times, to compromise when it comes to quality to save rendering time.

Ray tracing is a computer graphics technique meant for generating high-quality, photorealistic images by tracing the path of light through an image plane and simulating the optical effects of the objects in the scene [SH07]. A wide range of software products (open source, freeware and commercial) that support the ray tracing technique are available in the market including Autodesk Maya, Autodesk 3ds max, Blender and POV-Ray. Each of these products comes with a set of features, platform support and licensing terms. POV-Ray sits on top of applications that render 3D model using the ray tracing technique.

Researchers opt to utilize the parallel computing model to harness the power of multiple cores, processors and standalone machines in order to reduce the execution time of slow processes. Parallelism technique, mainly by means of multi-processors and multi-cores, and recently specialized graphics processors (GPUs) is being heavily studied to question its applicability to computer graphics. Theoretically, the ability to concurrently render different portions of complex 3D models on different processors/cores might generate the final images and scenes much faster.

Grid computing is a form of parallelization that could be utilized to render 3D images and films in short times. This objective can be efficiently accomplished by linking a (large) number of computers to simultaneously render different images of a complete scene(s). Forming grids out of traditional desktop machines and laptops available at enterprises is known as Desktop Grids or Enterprise Grids. Offices, departments, subsidiaries, universities and laboratories have a set of computers that are being used by their users to accomplish the work-related tasks.

In computer graphics, 3D rendering has two main scenarios, namely 1) Image Rendering; 2) Animation Rendering. In this paper, we propose a framework that utilizes desktop grids to speed up the rendering phase of the 3D graphics model using POV-Ray. The remainder of this paper is organized as follows: Introduction to the ray tracing technique and POV-Ray tool are explained in section 1. Section 2 introduces Grid Computing. Section 3 briefly discusses the technical details of Alchemi and how it is used to implement desktop grids. In section 4, we present our framework for rendering 3D models on desktop grids using POV-Ray. Last section concludes the paper and points out the work of the next paper.

1. Ray Tracing

1.1. What is Ray Tracing?

The idea of ray tracing, in simple terms, is to simulate the light source as it emits a light beam that eventually hits physical objects existing in its path so that our eyes could see them [Whi79]. In reality, when an object interrupts the path of a light ray, one or more of the following scenarios might occur: 1) the object may absorb a portion of that light ray, resulting in the loss of the light intensity; 2) the object may reflect part or all of the light; 3) if the surface is transparent, it refracts some of the light beam into itself in different directions; 4) the surface may absorb a portion of the light and fluorescently re-emits it at a longer wavelength color in a random direction.

In ray tracing, the quality of the generated images solely depends on the number of emitted light rays. That is to say, the more light rays are emitted, the higher the quality and the more photorealistic images can be generated. That is why in many cases, designers shoot multiple light rays for each single pixel in the image.

The ray tracing technique can efficiently and effectively simulate reflections and shadows that are difficult to simulate with other common techniques. However, the ray tracing technique is slow and computationally

expensive due to the extensive generation of light rays. This makes ray tracing more suitable for rendering images and films that usually take long times, and less suitable for real-time applications like video games.

1.2. The Persistence of Vision Raytracer (POV-Ray)

POV-Ray is one of the well-known and widely used ray tracing programs (http://www.povray.org/). POV-Ray is a freeware program that is being distributed under the POV-Ray license which allows the distribution of the program and its source code, but restricts commercial use and the creation of derivative works. The current official version of POV-Ray is 3.6 which is distributed in compiled formats for Windows, Linux and Macintosh. POV-Ray has a number of good features including support for textures, reflections, refractions, radiosity, fog, smoke, clouds; bump mapping; various types of light sources; and a library of ready-made objects, scenes and textures.

The creation of POV-Ray scenes can be made by writing scripts using the Scene Description Language (SDL) which comes as a core component of the program. SDL has a number of constructs that enables the programmers and the art designers to build complex scenes. This includes support for background colors, camera, lights, etc. Scene files are saved in ".pov" format which may refer to external libraries (".inc") and animation initialization files (".ini").

Interactions with POV-Ray usually occur through the provided graphical user interface (GUI) editor. In addition, command-line options are available so that users are able to write scripts and batches that can either be manually launched or automatically executed from within other programs.

1.3. Configuring POV-Ray

After downloading POV-Ray, implementers should perform the following steps:

1. Install POV-Ray on all the executor nodes.
2. Copy the "bin" folder to the same path on all the executor nodes so that the grid-enabled application can launch the "pvengine.exe" program from the command-line. This step is required since the installation wizard of POV-Ray does not allow the user to specify the installation path. Without performing this step, the grid-enabled application will not be able to locate and launch "pvengine.exe" on the executor nodes.
3. Configure POV-Ray as illustrated in Fig. 1. Skipping this step will cause the rendering phase to fail.
4. Create a shared folder and grant Read/Write permissions to users. Skipping this step will result in failure in saving the rendered work.

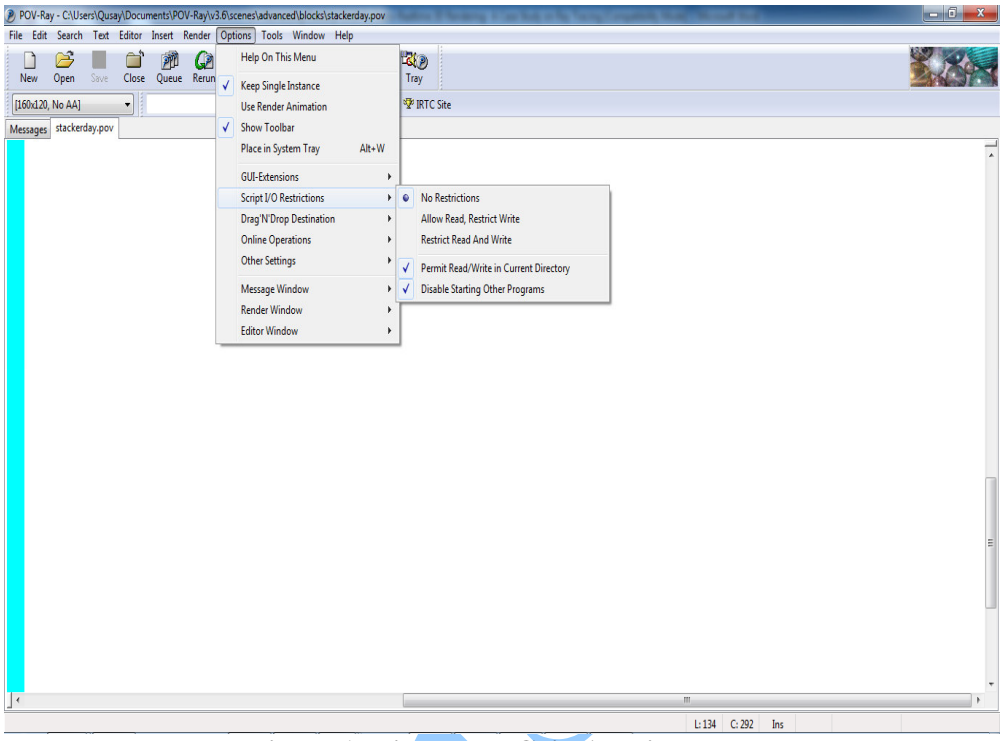

Figure 1. High Level Grid Architecture

2. Grid Computing

2.1. Definition and Architecture

Grid computing refers to the ability to combine, coordinate and share different computing resources in a scalable and dynamic manner to obtain powerful capabilities which can meet the complicated needs that are hard to cost-effectively fulfill by other means [FK04].

Supercomputers are usually utilized in cases where enormous resources are needed to solve very complex problems quickly and efficiently. Examples to these problems include deigning a new plane or vehicle, financial analysis and modeling, weather forecasting, risk analysis, urban planning, pharmaceutical engineering, film making and 3D rendering. However, most organizations, especially small-to-medium enterprises (SMEs) and non-profit organizations cannot afford such solutions. Grid computing allows organizations to have supercomputer-like performance but at a much lower

cost. This goal is accomplished by networking these resources in a way that allows end users to use them autonomously as if they were all coming from a single powerful computer.

As mentioned earlier, grid computing is one form of the parallel computing models where processing takes place concurrently on a number of nodes (also known as hosts). These nodes can be standalone machines, servers, clusters, supercomputers, or a combination of any of them.

The interaction between grid nodes can take different forms. The Master-worker parallel model is one of the commonly followed paradigms in grid computing. As its name denotes, this model is composed of two constituent parts: 1) master (also known as manager) which is a central node (or a cluster of nodes) responsible for dispatching work units; 2) workers (also known as executors) that work together to execute the grid jobs. Grid workers can either be full time workers dedicated to executing the incoming jobs, or non-dedicated where they only join the grid when they are idle to voluntarily execute the jobs and then leave when they are needed to perform traditional tasks.

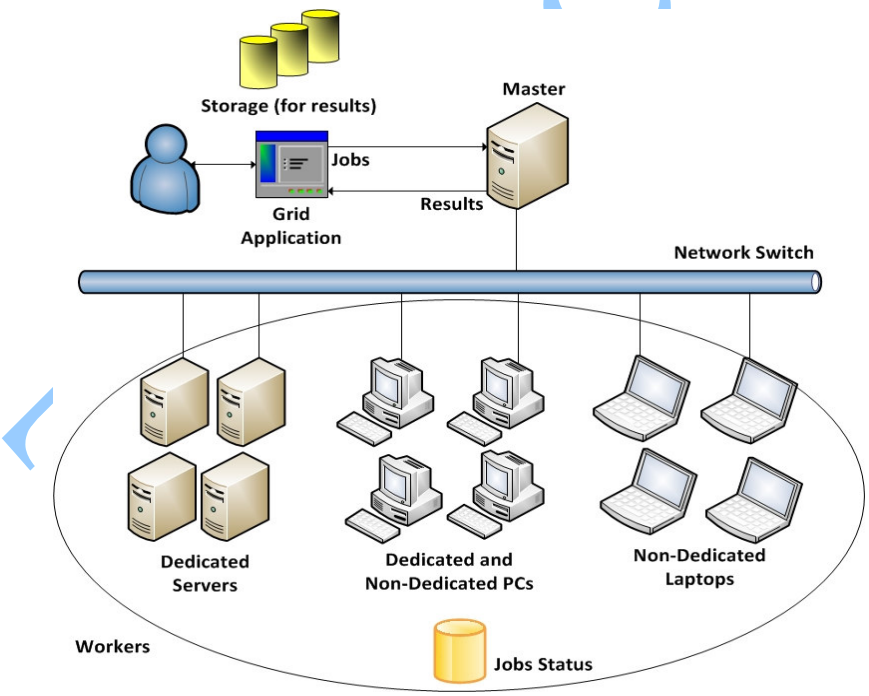

Figure 2. High Level Grid Architecture

As illustrated in Fig. 2, a user sends a complex job to the master node via a grid-enabled application. This client application is responsible for breaking a complex job into a number of smaller and simpler sub-jobs (also known as

subtasks). Then, the master allocates workers in order to assign the queued sub-jobs to them. The result of every completed sub-job is then sent back to the master in order to forward it back to the client. The client application logic determines whether to use the received results "as is" or to execute some post-processing operations such as formatting the results or combining them into a single output.

2.2. Building a Grid

To form any grid in general, implementers need the followings:

1. *Computer Network:* Forming a network is essential to link available computers together. LANs are usually used with desktop grids, but WANs are used when implementations span large geographic areas such as cities or even countries. This can be simply done using traditional network means such as switches cables, routers and internet connections (in case of physically distributed areas).
2. *Grid Middleware:* A special software that acts as an intermediary between the operating system (OS) installed on the grid nodes and the client application(s). The main functionality of grid middleware is to connect the available nodes (also known as workers or executors) and enable them to work together in a transparent manner. To be specific, grid middleware is responsible for receiving requests from clients; monitoring the connected nodes; preparing, scheduling and dispatching jobs to be processed by the available nodes; and finally receiving the result sets to return them back to the corresponding client(s). A number of grid middleware products that can be used by implementers are available. This includes, for example, Globus, Condor, BOINC, Gridbus, Alchemi, Utilify Aneka and Digipede. Each of these products is best suited for the scope/scale of the problems being solved and the environment on which they will be installed. For instance, BOINC was designed and developed based on the "volunteer computing" model, where nodes are globally distributed and their computing resources are directed to solve a specific problem; whereas, Alchemi was built to enable organizations to form grids by networking locally owned computers.
3. *Grid Framework:* A set of Application Programmable Interfaces (APIs) and tools that enable programmers and administrators to grid-enable applications and monitor the grid resources, respectively. Grid-enabling (or as sometimes referred to "gridifying") applications means building applications that can take advantage of the grid resources on hand. This term either refers to converting the existing applications (both readymade and tailored) that were written using the traditional (non-grid-based)

architectures, to grid-enabled applications, or writing brand-new applications. Usually, each grid middleware is packaged with a software development kit (SDK) that consists of its own set of APIs, tools, sample applications, test scenarios and manuals so that implementers can have integrated turnkey grid-computing solutions.

4. *Application:* On top of grid architecture, an application or more should be presented to allow end users to solve their computationally-intensive problems. Grid-enabled application should be written in a way that enables them to make efficient use of the available resources in the grid. That is to say, the application should include a logic that breaks complex jobs into a number of smaller and simpler jobs known as sub-jobs so that they can be distributed to the available computers to be solved in parallel. In addition, since the returned results are received from different computers in chunks, the grid-enabled application should be able to combine them into a meaningful manner as if they are all received as one single unit from one single computer. Failing to break complex jobs into a number of sub-jobs will cause the whole job to be sequentially executed on a single computer, which means losing the power of the existing grid resources. Similarly, failing to combine the result sets of the dispatched sub-jobs will end up with inconsistent, wrongly formatted or even meaningless outputs.

3. Desktop Grids Using Alchemi

Windows OS is the most prominent operating system around the world. Different versions of Widows OS are already installed on over 80% of the personal computers and corporate servers (http://www.w3schools.com/browsers/browsers_os.asp). Thus, deploying grids on a Windows environment is reasonable for both profit and non-profit enterprises.

.NET Framework (pronounced as "dot net") is the cutting-edge development technology offered by Microsoft for Windows environment. A number of grid technologies that are compatible with .NET environment are now available in the market. Some of these products are open-source such as Alchemi (http://www.cloudbus.org/~alchemi/), some are freeware such as Utilify (http://www.utilify.com/), and others are commercial such as Aneka (http://www.manjrasoft.com/products.html) and DigiPede (http://www.digipede.net/). In this work, Alchemi is used to build our grid-based POV-Ray rendering framework.

3.1. What is Alchemi

Alchemi is a grid-enabling middleware with a set of APIs developed by the GRIDS team at Melbourne University [LBRV05]. Alchemi is composed of three components: 1) Manager which acts as a grid master that manages the whole grid implementation process including the preparation and scheduling of the jobs and delivering results to the client application(s); 2) Executor which acts as a grid worker that actually process tasks; and 3) Dashboard that enables implementers to monitor the state and utilization of their grid(s), and the number of the running applications and executors.

3.2. Downloading and Installing Alchemi

The latest stable release of Alchemi is 1.0.6 which can be downloaded from the Alchemi page on SourceForge. Alchemi 1.06 works fine on Windows XP and Windows 7 and it comes with easy installation wizards. However, we have encountered some difficulties in installing or using Alchemi on Windows Vista. Readers can find the source code of more recent versions of Alchemi on SourgeForge, but we failed to implement a grid with any of them.

To make use of Alchemi, implementers should first install and configure it on their machines. The installation and configuration should take place on both the Manager and the Executor nodes. Usually, implementers install the Manager component on a single powerful node, and install the Executor component on the remaining number of machines.

3.3. Configuring Alchemi

After the successful installation of Alchemi, implementers start configuring and starting Manager and Executors. Fig. 3 illustrates the basic settings of both Alchemi Manager (on the left side) and Executor (on the right side). Configuring Alchemi Manager includes the definition of Port Number on which it will receive calls from the connected executors. After successfully configuring Alchemi Manager, implementers should be able start it with no problem.

The next step in the configuration phase is to configure Alchemi Executor on each machine. This includes the placement of information needed to successfully connect to the manager node such as Manager IP and Port Number, user credentials, whether the executor is dedicated or non-dedicated, and the storage technique (either MS SQL database or In-Memory) that will be used to log jobs is defined in this step.

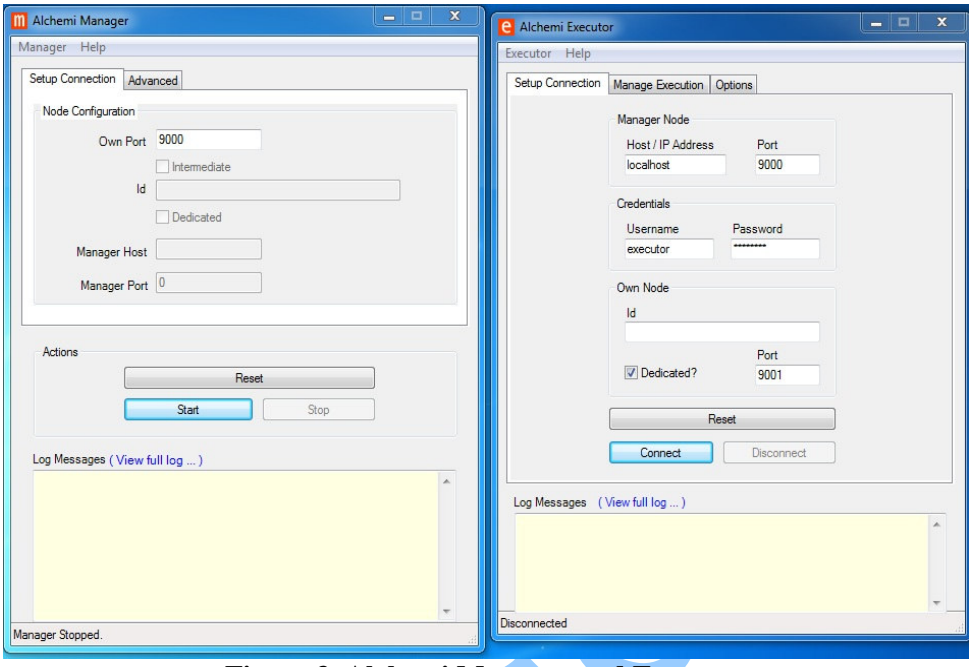

Figure 3. Alchemi Manager and Executor

3.4. Programming with Alchemi

Alchemi deploys the thread programming model where the computational process is subdivided into a set of smaller units known as threads [L+05]. The manager queues the created threads and schedule them according to the availability of the executors. If the number of queued threads is larger than that of the free executors, the manager assigns a number of jobs equivalent to the number of free executors, and when one of the assigned jobs has been completed, the manager sends a new job to the returning executor, and so on.

To write a class that is compatible with Alchemi, programmers should follow these steps:

1. Add a reference to the "Alchemi.Core" dynamic link library (.dll) in their client applications (or libraries) which contains all instructions needed to grid-enable the application using Alchemi. This library along with the user manual and configuration utilities can be found in the installation path.
2. Write a [Serializable] class (or more) that implements (i.e., inherits from) the "GThread" class. Marking said class with the [Serialzable] attribute is required to enable Alchemi to pass the class and included metadata between grid nodes using the "Remoting" technology offered by the .NET Framework. The "GThread" class represents a unit of work that can be

executed by Alchemi on one of the connected executors. Implementing the "GThread" class mandates that programmers implement the abstract method "Start()", otherwise the .NET Framework will not be able to compile the code. "Start()" method should contain the logic that will be executed by the Alchemi. Each execution using the "Start()" method represents an execution of one job on the grid. That is to say, programmers will be able to subdivide the core logic into a set of jobs by creating a number of instances of the "GThread" class which will be distributed by the manager to the available number of executors. Listing 1 illustrates an example to class that implements the "GThread" class.

3. As will be described later, programmers can get notified about the status and progress of the grid jobs via a number of events that is automatically fired by Alchemi.

Listing 1. An example to a class that implements the "GThread" class

```
using System;
using Alchemi.Core.Owner;
namespace PovRayRenderer
{
   [Serializable]
   public class RendererThread : GThread
   {

    //Implementation of GThread members
    public override void Start()
      {
       //Implementation Logic
      }
   }
}
```

4. Methodology

The 3D rendering of a large, high-resolution, deep image that contains many details and visual effects can take many hours to complete on a single computer. Similarly, 3D rendering of one short, high-resolution scene might take days to complete on a single computer. The ability to aggregate the computing power of the underutilized computers existing at enterprises can

reduce the time taken in the rendering phase significantly while increasing the return on investment (ROI) of these resources.

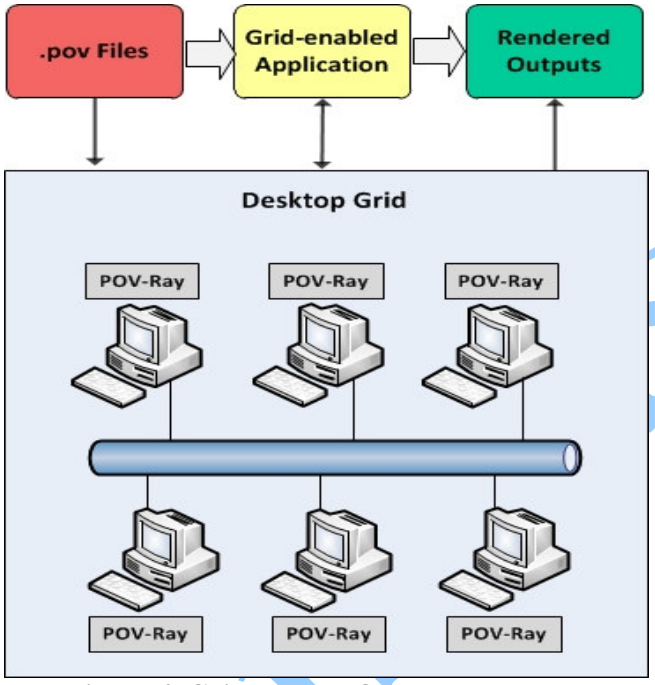

Figure 4. Grid-based POV-Ray Renderer

As illustrated in Fig. 4, our framework is simple in which:

1. A user (programmer or artist) submits the created ".pov" file (and any supplementary files such as libraries ".inc" or animation scenario ".ini") to the desktop grid either via the GUI layer laid on top of the grid-enabled application or by manually copying it to a known path. Regardless of the way used to pass the ".pov" and any additional files, they all must be set in a location that can be accessed by all grid nodes (e.g., network folder or database).
2. The grid-enabled application creates the grid jobs according to the settings specified by the user and then sends them to the grid.
3. The manager schedules the jobs and starts sending them to the free executors.
4. As mentioned earlier, POV-Ray must be pre-installed on each executor. When an executor receives a rendering job from the manager, it launches POV-Ray in a separate process and passes the arguments accompanied by the job.

5. POV-Ray renders the job and when it is finished, the launched process exits itself and the executor returns the results back to the manager.
6. The manager then forwards back the rendered outputs to the grid-enabled application and fires "ThreadFinished" if successfully completed or "ThreadFailed" if an error occurred. The rendered output should be saved on a shared media which all executors can have access to.
7. When all jobs are finished, the manager fires the "ApplicationFinished" event. This event enables programmers to post-process results and/or release resources when the whole application is finished. Post processing can include for example, creating a movie out of the rendered frames, or combining various strides to construct a complete image.

Conclusion

While computers are getting more powerful, most of them usually are set idle in enterprises and their capabilities are rarely made use of. Performance counters provided by both the OSs and third-party add-ins show that about 20 - 40% of the CPUs' power are only being used while performing traditional tasks such as playing music, word processing, chatting and Internet surfing. Contrarily, 3D rendering is a time-consuming and computationally-intensive process that often causes powerful CPUs to peak while running.

This work presented a framework by which artists could render their 3D models at a fraction of the time without incurring additional costs. The framework has utilized Alchemi, a grid enabling middleware, and .NET Framework, the leading programming environment on Windows, to combine the traditional workstations available at enterprises in order to form powerful processing units that imitate the performance of supercomputers. While the work was customized to POV-Ray, it can be easily generalized to work with other modeling and rendering technologies.

The implementation details of the proposed framework will be presented in the next paper. This will include a parallel, grid-based rendering of both image and video rendering [RHH11].

Acknowledgment

The authors would like to thank Rana Sherif for her generous assistance in editing this work.

References

[FK04] **I. Foster, C. Kesselman -** *The Grid 2, Second Edition: Blueprint for a New Computing Infrastructure*, Elsevier, 2004

[L+05] **A. Luther, R. Buyya, R. Ranjan, S. Venugopal -** *Alchemi: A .NET-Based Enterprise Grid Computing System*, Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), June 2005, Las Vegas, USA.

[RHH11] **A. M. Riad, A. E. Hassan, Q. F. Hassan -** *Implementation and Evaluation of POV-Ray on Desktop Grids: Parallel Rendering of 3D Images and Animations*, Anale. Seria Informatica. Vol. IX fasc. 2 – 2011.

[SH07] **K. Suffern, H. H. Hu -** *Ray Tracing from the Ground Up*, September 2007.

[Whi79] **T. Whitted -** *An improved illumination model for shaded display*, Proceedings of the 6th annual conference on Computer graphics and interactive techniques, 1979.