Cluster Based Modeling and Graphical Visualization of Interactive Large Spatial Data

Cristinel Mihai Mocan, Dorian Gorgan
Computer Science Department
Technical University of Cluj-Napoca
Str. G. Baritiu 28, 400027, Cluj-Napoca, Romania
{cristi.mocan, dorian.gorgan}@cs.utcluj.ro

Abstract - Rendering and interacting with high-resolution geographical data sets and complex models of virtual geographical space involves high power computation resources in networking environments. The solution of integrating graphics rendering engine applications in cluster based architecture and Grid infrastructure is the main concern of the research reported by this paper. The performance of load balancing is evaluated for various cluster configurations by considering different combinations of distributed rendering algorithms over the graphics cluster and spatial data models. The research studies and experiments as well the flexibility related with the mesh data formats accepted by the graphics renderer, user interaction techniques with complex scenes in the context of graphics cluster rendering, and solutions for data streaming and unit frame encoding. A few use cases of visualization of the virtual geographical model exemplify the achievements.

I. INTRODUCTION

In case of complex 3D models a single GPU cannot offer an adequate performance although the rendering devices performance has been the subject of a surge related to this aspect.

Our goal is to allow the user to view and interact remotely with complex scenes on his computer using a cluster based rendering system. The challenge is how to integrate a flexible object-oriented graphics rendering engine like scene-oriented with a parallel rendering framework in order to develop scalable graphics applications for a wide range of systems ranging from large distributed visualization clusters and multi-processor multi-pipe graphics systems to single-processor single-pipe desktop machines. Related to this challenge we focused on finding an object-oriented graphics rendering engine that produces applications utilizing hardware-accelerated 3D graphics. Also we look for a toolkit based on OpenGL which provides an application programming interface to develop scalable graphics applications for our graphics cluster. Increasing multi-core processor and cluster-based parallelism and the improvements concerning the CPU and GPU performances demand for flexible and scalable parallel rendering solutions that can exploit multi-pipe hardware accelerated graphics. In this situation, the scalable rendering systems are essential to cope with the rapid growth of data sets in order to achieve interactive visualization.

The task of integrating an object-oriented graphics rendering engine with a scalable parallel rendering framework is even more difficult if it should be generic to support various types of data and visualization applications and at the same time to work efficiently on a cluster with distributed graphics cards.

From the interactivity point of view the solution is to allow the user to interact with the complex scenes. Using the object-oriented graphics rendering engine the user can add a camera in the application. In this situation the user is able to navigate throughout the scene interacting by the input devices - by keyboard keys to move the camera and by mouse for camera rotation.

The next section highlights a few related works. Section 3 explains why and how we integrate object-oriented graphics rendering engine with the parallel rendering framework. Section 4 presents the system architecture and Section 5 describes the load-balancing rendering strategies used in our experiments. The experimental scenarios and the evaluation reports are described in Section 6. The last section concludes on the proposed solution and sketches the future work directions.

II. RELATED WORKS

A great part of work has addressed issues recently in accessing, representing, and manipulating large data sets. The previous work can be summarizing as follows.

The paper [1] presents a few experiments on the solutions supported by the Chromium graphics cluster to provide fast distributed processing and remote visualization. It evaluates the performance for various cluster configurations and spatial data models. The research concerns as well with accessing the cluster based processing by web applications through grid and web services.

A solution of combining graphics cluster based computation, Grid infrastructure and Web applications appears in paper [2]. The paper explores and experiments the optimal architecture for remote graphics visualization considering different combinations of the sort-first and sort-last distributed rendering algorithms over the Equalizer graphics cluster.

Concerning the parallel rendering we found out that Equalizer [3] supports multi-view and scalable rendering, resource management, and planning of the transparent rendering layer. It enables application developers to configure their applications to take full advantage of multi-GPU workstations. In this case the applications provide better performance and allow visualizing complex data sets.

Equalizer and Chromium [4] solve similar problems by enabling applications to use multiple GPU’s, but have disjunctive use cases and characteristics. In our
experiments we have chosen to work using the parallel rendering framework because scalability, flexibility and compatibility are mainly required in our graphics cluster.

On the other hand, in order to build complex oriented-scenes applications we compared some graphics engines like [5], [6], [7] and [8]. The conclusion is that Ogre includes the features available in other graphics engines as add-ons. It offers more freedom to choose the components you want and has much better geometry and material formats and handling. Also the design is better, faster, and is much more feature rich. Irrlicht has mostly obsolete file formats that were never designed for real-time graphics and are extremely limited, often forcing you to use several separate formats to get the specific material and features you need. Finally, I consider using object-oriented graphics rendering engine because it uses a flexible class hierarchy that allows us to design plug-ins to specialize the scene organization approach taken to allow us to make any kind of scene. Therefore, the object-oriented graphics rendering applications can be integrated with our parallel rendering framework. Concerning the remote and collaborative visualization, the openVTISSAR toolkit [9] sends the real-time generated visualizations.

III. OBJECT-ORIENTED GRAPHICS RENDERING ENGINE INTEGRATION

In order to render and interact with high-resolution geographical data sets and complex models of virtual geographical space, in the beginning the challenge was to find an object-oriented graphics rendering engine to make it easier and more intuitive for us to produce applications utilizing hardware – accelerated 3D graphics. The class library we found in object-oriented graphics rendering engine abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes. It is very important to remark that this library uses a flexible class hierarchy allowing us to design plug-ins to specialize the scene organization approach taken to allow us to make any kind of complex scene we like.

On the other hand, from the point of view of the graphical cluster we modified the Equalizer framework which is an open source parallel rendering framework in order to solve the integration with the graphics rendering engine. The applications based on the framework we obtained after the modifications provide better performance and allow visualizing complex data sets.

The solution of integrating graphics rendering engine applications in cluster based architecture using the parallel rendering framework is the main concern of the research reported by this paper.

IV. gcIVis ARCHITECTURE

The proposed architecture for the cluster based visualization system uses the client-server model for the rendering part (Fig.1).

The main gcIVis architecture components are the following: the visualization servers, the rendering clients and the resource manager component. The communication and the user interaction use a resource manager and a notification model.

A. Rendering Nodes and Server

At the server and the rendering client level we used the modified parallel rendering framework based on Equalizer.

For our experiments we used multiple computers that were linked together in order to share computational workload. The requests initiated from the user are managed by and distributed among all the standalone computers to form a graphics cluster. In designing the graphics clusters we used different computing resources depending on the conducted experiment.

Each of the rendering client has a module which is responsible to execute the rendering tasks sent by the server. In conclusion, a rendering node application runs on each rendering client. After each rendering cycle, the result is sent to the composer node. In order to achieve a good performance, the result data has to be compressed. For this purpose we use a compression plugin system supported by Equalizer. Currently the data is compressed using the RLE (Run Length Encoding) method. This is not the best option, a Huffman coding could achieve a better compression ratio. Another criteria for selecting other algorithm is execution time. From this point of view, the RLE execution time is linear with the raw data size.

![Fig. 1. gcIVis architecture based on Equalizer middleware](image-url)
The Rendering Server and the Composer application are running on the rendering subsystem’s server. The server is responsible for managing the rendering nodes, transmitting messages to nodes and object distribution. The configuration of the rendering system is described in a configuration file [10]. The Composer module objective concerns on composing the final frame, compressing it, and streaming a continuous video stream to the client. For video compression we used the free library, which is a part of the FFmpeg project [11]. The communication between the streamer process and the rendering system uses pipe files. The composer node creates a child process, using the streamer binary and redirects its standard input to a pipe file. The streamer will permanently read the standard input and stream the data to the destination. The video streaming is made on a constant frame-rate, so the streamer process has to be fed continuously, even if the rendering system’s performance is lower as the desired frame-rate or it is not constant. This problem was fixed by a simple algorithm, which skips some of the frames, if the rendering FPS (Frames per Second) is greater than the streaming FPS or duplicates frames in the other case.

The networking layer provides a peer-to-peer communication infrastructure. It is used in order to communicate between the application node, the server and the render clients. This layer can also be used by applications to implement distributed processing independently or complementary to the core Equalizer functionality. It provides layered functionality, which means higher level functionality for the programmer. In our graphics cluster network layer the communication between nodes is accomplished using packets. There is a base class that allows the registration of a packets with a dispatch queue and an invocation method. Each packet has a command identifier, which is used to identify the registered queue and method. We can say that the node is the abstraction of a process in our graphics cluster. Using connections the nodes communicate with each other.

Based on the Universally Unique Identifier [12] standard each node has a unique identifier. For instance, in order to query connection information to connect to the node we use the identifier to address nodes.

B. Node Manager and Broker component

In order to achieve the desired experiments from this research we developed a node manager application. Using it we could enable or disable certain nodes in our graphics clusters depending on how many we need in each experiment.

We use this component for the cluster administration. By this tool the administrator may register new nodes that can be disabled or removed later from the system. It is important to notify that the nodes can be Server or Client nodes. This is a desktop application that connects to the database installed on the server and reads/modifies the database table which stores the information about the registered nodes.

The Node-Manager offers application file deployment. That means the application files are copied using Secure Copy Protocol before a new node is added in the system. In order to make it work, the host on which the Node-Manager is installed needs to have a password less SSH connection configured to the new node.

The administrator interest is to have access to a friendly user interface through which a new rendering node can be added to the rendering system. It is important to have access to the database and as an alternative scenario we assume the node could not be added successfully because of different reasons, like connection less or password less SSH connection errors.

On the other hand, the broker component receives requests from users and, depending on the rendering strategies and parameters it fetches the visualization to a rendering server. The rendering clients are receiving the rendering parameters from the rendering server together with the graphical scene.

Based on visualization requirements, the broker transparently selects the most suitable cluster for rendering. The cluster level resource manager is the local dispatcher that selects the visualization server in the cluster, and fetches the input data. This system level manages the existing clusters at a higher level.

Depending on the rendering attributes selected by the user, the visualizing service selects the appropriate read back component. The visualizing system provides three features: creation of video streaming visible in the user interface; image, when the cluster renders only one image frame; and video sequence, which is actually a movie as a set of image frames.

The client application is using the UI (User Interface) which allows the user to create a complex scene using the object-oriented graphics rendering engine library which uses a flexible class hierarchy. The UI allows the user to make any kind of complex scene we like with some limitations regarding the data formats accepted for the loaded objects. Because we focused on complex models, in this moment the user application accepts only ply (Polygon File Format) models. Using the camera feature from the UI, the user controls and manipulates the visualization scene. It supports the user interaction with the virtual scene, mainly concerning with camera manipulation and interaction techniques to individual scene objects. The UI component receives commands from the user and forwards them to the rendering nodes using a communication channel.

V. LOAD BALANCING – RENDERING STRATEGIES

Using the load-balancing we can increase the performance of our graphics cluster systems, because this results in balanced computational work among different machines.

For optimal rendering performance sort-first and sort-last compounds often need load-balancing while pixel and stereo compounds are naturally load-balanced.

Some applications do not support dynamic updates of the database range, and therefore cannot be used with sort-last load-balancing. Using a sort-first or sort-last load-balancer will adjust the sort-first split or database range automatically each frame.

There are three modes in which the sort-first load-balancer works: sort-first using tiles, horizontal using rows and vertical using columns.

Sort-first load-balancing increases the frame-rate over a static decomposition in virtually all cases. It is very important to remark that we obtained the best performance if the application data is relatively uniformly distributed in
screen-space. In order to fine-tune the algorithm we can use a damping parameter.

From the point of view of the sort-last load-balancing it is known that this rendering strategy is beneficial for applications which cannot precisely predict the load for their scene data. For example when the data is non-uniform.

On the other hand a static sort-last decomposition typically results in a better performance if we have a volume rendering example with uniform data. Considering that a segment represents a single display (a projector or monitor) and that it references a channel (has a name, viewport and frustum) is important that each segment of a multi-display system has a different rendering load and it depends on the data structure and model position. The channel referenced by the segment defines the output channel. In the case we use a static assignment of resources to segments the overall performance is determined by the segment with the biggest load. The solution is presented in our experiments related to load-balancing performances.

Regarding the wrong balancing, some configurations require a smoothing of the frame-rate at the destination channel, otherwise the frame-rate will become periodically faster and slower. Using a frame-rate we will smooth the swap buffer rate on the destination window for optimal user experience.

On the other hand the dynamic frame resolution trades rendering performance for visual quality. In order to keep the frame-rate constant the rendering for a channel is done at a different resolution than the native channel resolution. The solution is that dynamic frame resolution adjusts the zoom of a channel, based on the target and current frame-rate. For instance, volume rendering and ray-tracing which are fill-rate bound applications are usually using dynamic frame resolution. For example, the key for obtaining ten frames per second instead of five is that the model is rendered at a lower resolution and upsampled to the native resolution for display.

Concerning the rendering quality we have to note that it is slightly degraded, while the rendering performance remains interactive. It renders a full-resolution view when the application is idle. The dynamic frame resolution will also upscale the resolution if the parameters allow for it. In conclusion, it does not have a limitation in order to downsampling the rendering resolution. Beside this upsampled rendering which will down-sample the result for display, provides dynamic anti-aliasing at a constant frame-rate.

VI. EXPERIMENTS

The architecture we proposed has been experimented on two local graphics clusters.

The computing resources used for the first one consists of six Pentium 4 systems with 1 GB RAM memory, GeForce 8800 640MB graphic card, and running on Windows XP SP3. This graphics cluster was used for the first three experiments described in detail in the following paragraphs.

In terms of scalability, in the last experiment we used the second graphics cluster consisting of eleven Intel® Core™ 2 Duo with 2 GB RAM memory, GeForce 9600GT graphic card, and running on Windows XP SP2.

For evaluating the functional level of our architecture we have defined a set of test scenarios.

A. Scene-Oriented integration into Graphics Cluster

The aim is to evaluate the integration of all architectural components by using a very general use case scenario. For building complex scenes, we used an object-oriented graphic engine [5] which we integrated in the parallel rendering framework based on Equalizer middleware. This use case refers to the graphics cluster based visualization and user interaction in a complex 3D scene.

For this experiment we used a graphics cluster consisting of two rendering clients and one server of which task concerns with scene data distribution over the rendering clients and the composition of the result received from each node.

Through the client application, the user builds up the graphical scene and sets the rendering parameters from the user interface. An example of the application we built is shown in (Fig. 2). In this scenario, the user sets the positions for each of the selected scene model - the ground material and some complex models (a house and different types of trees). In order to create this complex scene, the client application used the scene management and the resource management classes from the library used in object-oriented graphics rendering engine.

In this use case the application load several ply complex models to hardware buffer and then render the scene with texture shadow. After the user sets the rendering parameters and configures the graphics cluster by the manager, the application can start the parallel rendering using the graphics cluster. It is a scene-oriented application that load ply files as mesh data and computes textured shadow for each object in real-time.

Regarding the graphical rendering, the UI allows the user to choose between these configuration modes of the system: multi-window, multi-channel or graphics cluster.

The user interaction is supported by keyboard, in order to move the camera, and by mouse for camera rotation.

Using our work for the integration of the object-oriented graphics rendering engine with the Equalizer framework, the user has the possibility to create the camera and to set its position.
By default we set the camera at 500 units on Z direction, and looking back toward –Z. Then the viewports are created as well. In our use case, we map the viewport over entire window.

Concerning our experiment with the object-oriented graphics rendering engine integration we can conclude that the visualization results show that the performance increases only for complex models.

For simple models, the performance declines by increasing rendering nodes. On the other hand, because of the object-oriented graphics rendering engine limitation, thread safeness is poor. Each node can contain only one GPU, which means we must change the Equalizer configuration file (two-window using sort-first strategy) to use one pipe that contains two windows.

B. Graphics Cluster visualization performance using load-balancing strategies

The aim of our next experiments reported in this research concerns on evaluating the impact of scene complexity, image dimension, and rendering method using different load-balancing strategies on the visualization performance.

B.1. Resources assignments for optimal performances

In the next experiment we used our first graphics cluster having all the computing resources available. Concerning the segment definition from chapter V, it is important that each segment of a multi-display system has a different rendering load and it depends on the data structure and model position. When using a static assignment of resources to segments, the overall performance is determined by the segment with the biggest load.

The solution is to analyze the load of all segments and adjusts the resource usage each frame. It equalizes the load on all segments of a view. This process is illustrated in (Fig. 3).

In this experiment we used a static assignment of resources to display segments on the left side. The right-hand segment has a higher load than the left-hand segment, causing sub-optimal performance. On the left side the configuration is using a view which assigns two GPU’s to the left segment and four GPU’s to the right segment, which leads to optimal performance for this model and camera position.

B.2. A relevant advantage in using load-balancing

The third experiment use the computing resources from our first local graphics cluster described in the beginning of this chapter and have four active rendering clients. The number of frames per second (fps) was used as the measured parameter in order to evaluate the performances.

First of all we compare and evaluate the visualization performances between sort-first rendering strategy and load-balanced sort-first rendering using objects centered in the middle of the screen. The differences between these modes are quite insignificant. For instance, if we upload a complex model without load-balancing we obtain 13 FPS. On the other hand, using the load-balancing, we obtain 17 FPS. In (Fig. 4) we show a demonstration regarding these results. For these measurements we load a complex model that has 10,000,000 faces and a total size of 183 MB.

From the measurements point of view, we have to conclude that the advantages in using load-balancing are relevant in the case the loaded model is covering a small part of the screen-space. In this case we obtained 17 FPS without using load-balancing and 25 FPS using it.

B.3. Frame computation by the rendering algorithms

For the last experiment we have used our second graphics cluster (one visualization server and ten rendering clients) in order to find out how much we can increase the number of rendering nodes and how this may influence the performances using the rendering algorithms with load-balancing on different resolutions. The test variables are the number of rendering nodes, the scene complexity (in term of number of triangles) and the rendering resolution.

Our measurements results presented in (Table 1) were obtained by running the graphics cluster for each loaded model on different resolutions. For these measurements we used 3 models with different complexity level. In terms of rendering algorithms we used the sort-first, sort-last and D Plex decomposition strategies to compare the performances related to load-balancing improvements.

D Plex decomposition requires multiple frames to be
### TABLE I
FRAME COMPUTATION BY THE RENDERING ALGORITHMS

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Model complexity level</th>
</tr>
</thead>
<tbody>
<tr>
<td>320 x 240</td>
<td>Small</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Large</td>
</tr>
<tr>
<td>640 x 480</td>
<td>Small</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Large</td>
</tr>
</tbody>
</table>

Legend:
- Sort-first without load-balancing
- Sort-first with load-balancing
- Sort-last without load-balancing
- Sort-last with load-balancing
- Dplex

rendered concurrently. Dplex mode assigns full frame rendering for each rendering client. This mode is known as the alternate frame rendering. It provides a very good scalability, but the disadvantage concerns the rendering system delay. In our experiment the Dplex is configured using a period and phase for each rendering node.

The network traffic and the rendering nodes load are variables that required our attention. The server node synchronizes the rendered frames at the composition stage. In this context the frame rate is influenced by the load of the rendering client nodes.

The results show that the performance increases only for complex models. For simple models, the performance declines with increasing rendering nodes. We can conclude that in the case of the sort-first strategy, the scalability may be applied, but in the case of sort-last, the network traffic cost is increasing with the number of rendering nodes. Concerning our measurements the advantages in using the load-balancing are relevant in the case the loaded scenes are complex and the rendering algorithm is sort-first.

### IV. CONCLUSIONS

The research has developed and experimented the main functionality and distributed architecture providing the modeling and graphical visualization of large spatial data space by integrating graphics rendering engine applications in cluster based architecture. Nevertheless the performance already achieved using load-balancing strategies are quite far of the required ones. The future research will mainly concern with performance enhancement by graphics cluster configuration, streaming process improvements, communication and connectivity between graphics cluster nodes, rendering algorithm optimization and more data formats accepted by user interface.

### ACKNOWLEDGMENT

This research is supported by PRODOC (Doctoral Studies Development in Advanced Technologies) Project, http://prodoc.utcluj.ro.

### REFERENCES