Remote Graphical Visualization of Large Interactive Spatial Data

ComplexHPC Spring School 2011
International ComplexHPC Challenge

Cristinel Mihai Mocan
Computer Science Department
Technical University of Cluj-Napoca
cristi.mocan@cs.utcluj.ro
Outline

- Domain
- Use Cases
- Objectives
- System design – gVis Architecture
- Visualization workflow
- Rendering Components
- Load balancing - Rendering Strategies
- Multi-user interaction
- Experiments
- Conclusions
- Future work
Remote Graphical Visualization of Large Interactive Spatial Data

• Research work in the following fields:
  
  • High performance computing
  • Graphics cluster based processing and visualization
  • Computer graphics
Objectives

- The main goal:

- to allow the user to **view and interact remotely with complex scenes** on his computer using a **cluster based architecture and Grid infrastructure**.
Objectives

- To use the power of **multi-GPU systems** and **visualization clusters**

To run

different complex **3D Virtual Geographical Space (VGS)** scenarios

aiming at the **maximization of the GPU utilization**.
Objectives

- GPU Sharing
- Multiple Remote Users per GPU using Virtual Network Computing
Objectives

- Evaluate the performance of load balancing for various configurations by considering different combinations of distributed rendering algorithms over the graphics cluster and spatial data models.

Hybrid algorithms based on:
- Sort-first and Sort-last rendering strategies
System design

- **Software responsibilities:**
  - **Cluster Manager specialized on Visualization Resources**
    - we can use one or more nodes with GPUs
      - as a shared remote visualization farm
      - to run serial or parallel GPU enabled apps
      - to drive display walls
    - enhanced to support GPU sharing
  - more than one remote visualization session could be hosted off a single GPU.
System design

- **Software responsibilities:**
  - Visualization software

  Challenge?
  object-oriented graphics rendering engine + parallel rendering framework

  \[ \Rightarrow \text{to develop scalable graphics applications}
  \]
  for a wide range of systems
System design

• Software responsibilities:
  • Visualization software

In our experiments: **Equalizer framework**

Why?

*Scalability*

*Flexibility*

*Compatibility*

=> are mainly required for multi-user support.
System design

- **Software responsibilities:**
  - Visualization software

**gVis Architecture - Components**

Based on Equalizer middleware
Visualization workflow - 1

• The communication and the user interaction use a broker and a notification model.

• The broker component
  ▪ receives requests from users
  ▪ depending on the rendering strategies and parameters, it fetches the visualization to a rendering server.
Visualization workflow - 2

- The rendering clients
  - receiving the rendering parameters from the rendering server together with the graphical scene.
Visualization workflow - 3

• The encoder component
  ▪ fetches the rendered frames to the streaming server.

• The client application
  ▪ connects to a streaming channel and, using the UI, controls and manipulates the visualization scene (camera parameters, individual object parameters etc.).
Visualization workflow - 4

- The streaming server
  - creates streaming channels to which the clients are connecting.
  - receives the rendered frames from the composition node or the server node.
The user interface component

- supports the user interaction with the virtual scene, mainly concerning with camera manipulation and interaction techniques to individual scene objects.

- receives commands from the user and forwards them to the rendering nodes through a communication channel.
• Depending on the rendering attributes selected by the user, the visualizing service selects the appropriate read back component.
Visualization workflow - 7

- The visualizing system provides three features:
  - creation of video streaming visible in a web-based application;
  - image, when the cluster renders only one image frame;
  - video sequence, which is actually a movie as a set of image frames.
Object-Oriented Graphics Rendering Engine Integration

- **Ogre:**
  
  The class library
  - abstracts all the details of using the underlying system libraries like Direct3D and OpenGL
  - provides an interface based on world objects and other intuitive classes.

**Graphical cluster:**

We modified the Equalizer framework

(open source parallel rendering framework)

=> to solve the integration with the graphics rendering engine.
gVis Architecture-Multi-user interaction

- Support for different multi-user interaction techniques.
  - **master-slave** visualization model
    - Example: teaching activities
  - **client-server** visualization model
    - the system creates different rendering threads for every connected clients.

=> every single user can select:
  - a different visualizing scene
  - rendering strategies
  - different visualization parameters
Experiments

- **Evaluate:**
  - the impact of scene complexity
  - image dimension
  - rendering method

on the performance of remote visualization

- Measured parameter: **the number of frames per second (fps)**
Experiments

- Use Case: 1

3 different models

<table>
<thead>
<tr>
<th>ID</th>
<th>Model name</th>
<th>No. Points</th>
<th>No. faces</th>
<th>Dimension (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Laura50k</td>
<td>27861</td>
<td>49954</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>male01.ply</td>
<td>665.847</td>
<td>1.331.575</td>
<td>28.5</td>
</tr>
<tr>
<td>3</td>
<td>xyzrgb_statuette.ply</td>
<td>4.999.996</td>
<td>10.000.000</td>
<td>183</td>
</tr>
</tbody>
</table>
Experiments

- Client Application
  - Example:

View-Sharing
- Public/private session
- View session
Experiments

- **Client Application:**

  ![Client Application]

- **Public Session**

  ![Public Session]

- **View Public Session**

  ![View Public Session]

- **Master - Slave – example: for teaching activity**

  ![Master-Slave Example]
Experiments

- Experimental results:
  - Performance gain:
    - Medium resolution
    - High complexity model
  - System bottleneck – inter-node communication
    - Better compression
    - Faster network
      (currently 1gbit)
  - System advantage
    - Easy to use remote rendering system
  - System disadvantage
    - Latency ~ 1.5 sec
Experiments

- **Use Case:** 2
  - 3 different graphical scenes

**3D Virtual Geographical Space Scenarios**

- The Number of Faces for 3 different Maps

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>189,192</td>
</tr>
<tr>
<td>Medium</td>
<td>1,158,801</td>
</tr>
<tr>
<td>High</td>
<td>5,321,025</td>
</tr>
</tbody>
</table>
Experiments

- Performance Testing
Experiments

- **Performance Testing**
- **Frame Computation by the Sort-First Algorithm**

<table>
<thead>
<tr>
<th>Scene complexity</th>
<th>Nodes</th>
<th>256 x 256 pixels</th>
<th>512 x 512 pixels</th>
<th>1024 x 1024 pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low – small map</td>
<td>2</td>
<td>47</td>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>40</td>
<td>28</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>39</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>Medium – medium map</td>
<td>2</td>
<td>28</td>
<td>18</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>20</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>23</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>High – large map</td>
<td>2</td>
<td>14</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>11</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Best performance** related with image resolution and scene complexity obtained by using two or three rendering nodes and a middle image resolution.
Experiments

- Load balancing performance

Four nodes graphics cluster running a sort-first rendering strategy using load balancing:

a) Full object visibility

b) Partial object visibility

The measured parameter for evaluation = No. of frames per second

Load-balancing for two segments using 6 GPU’s:
Experiments

- **Use Case:** Scalable rendering
  - **Example 1:** Volume rendering

*Volume (sort-last) decomposition*

- allows to visualize data sets which do not fit on a single GPU
- The individual GPU only need to render a sub-volume of the whole data set.
Experiments

- **Use Case:** 3  
  **Scalable rendering**

- **Example 1:** Volume rendering
Experiments

- **Use Case:** 3  Scalable rendering
- **Example 1:** Volume rendering
Experiments

- **Use Case:** 3  Scalable rendering
- **Example 2:** Polygonal rendering
Conclusions

- **Use Case 3**

- Polygonal data sets have the **disadvantage** that the database recomposition is twice as expensive, since both color and depth information is processed.

- Furthermore, **load balancing** is harder compared to volume rendering since the data is less uniform.

- The hardware limits the rendering to 7.25 fps, half of the volume rendering performance.
Conclusions

- **Use Case 3**

  - Screen-space decomposition again suffers performance due to the fact that the whole model has to be loaded on each node.

  - This polygonal rendering benchmark is much less fill-bound than the volume rendering benchmark.
Conclusions & Future works

- The achieved system has been proved to be a very promising solution for scalability issues that involve multi-user and multi models working sessions.

- The experimental results obtained so far indicate that the reachable speedup strongly depends on the scene.

  next research efforts will be mainly focused on:
  - performance enhancement by graphics cluster configuration
  - rendering algorithm optimization
  - virtual space modeling and distributed processing
  - streaming and user interaction
Thank you for attention!

Cristinel Mihai Mocan
Computer Science Department
Technical University of Cluj-Napoca
cristi.mocan@cs.utcluj.ro