Eurographics Symposium on Parallel Graphics and Visualization (2010)
J. Ahrens, K. Debattista, and R. Pajarola (Editors)

Asynchronous Parallel Reliefboard Computation for Scene Object Approximation

Tim Süß[†] and Claudius Jähn[‡] and Matthias Fischer

University of Paderborn

Abstract

We present a parallel algorithm for the rendering of complex three-dimensional scenes. The algorithm runs across heterogeneous architectures of PC-clusters consisting of a visualization-node, equipped with a powerful graphics adapter, and cluster nodes requiring weaker graphics capabilities only. The visualization-node renders a mixture of scene objects and simplified meshes (Reliefboards). The cluster nodes assist the visualization-node by asynchronous computing of Reliefboards, which are used to replace and render distant parts of the scene. Our algorithm is capable of gaining significant speedups if the cluster's nodes provide weak graphics adapters only. We trade the number of cluster nodes off the scene objects' image quality.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.1]: Parallel processing—Computer Graphics [I.3.2]: Distributed/network graphics—Computer Graphics [I.3.3]: Display algorithms—

1. Introduction

When planning cities, machineries, buildings, and factories the complexity of three-dimensional scenes for their visualization increases continuously. The scenes result from CAD data and consist of more and more objects each, composed of an increasing number of polygons, caused by lots of details. For design review, layout check and other purposes, a real-time walk-through with high frame-rates (at least 10 fps [MH99]) is necessary. Often, the number of primitives in such scenes is so high that a single graphics adapter is not capable to render the whole geometry in real time on its own.

One common approach to face this problem is to compute the frames in parallel. Several approaches and architectures for parallel rendering have been suggested [MCEF94]. PC-clusters [SFL01,SFLS00] intended for parallel rendering offer a lot of distributed memory and frames can be computed by multiple graphics adapters.

However, most PC-clusters do not provide graphics accelerators or their 3D performance is weak if they are intended mainly for other applications, e.g. scientific computations. For example, the PADS and TeraPort PC-clusters at the University of Chicago do not provide OpenGL accelerating hardware, just as the JUGENE PC-cluster of the Jülich Supercomputing Center does not either. Some systems, the Paderborn Center of Parallel Computing's (PC^2) Arminius PC-cluster for example, are equipped with weak OpenGL accelerating graphics adapters. If such clusters should be deployed for visualizations purposes too, an upgrade of the complete PC-cluster is rarely done: Upgrading graphics adapters in a PC-cluster is expensive and problematic because high-performance graphics adapters produce a lot of heat. The PC-clusters' energy consumption and temperature control are a delicate topic for every provider. Sometimes, upgrading a few dedicated nodes can be done with much less effort. Furthermore, PC-clusters unspecialized in graphics rendering, can hardly be used for modern rendering algorithms, which require newer shader models or other new extensions.
To overcome this problem, we develop an algorithm that needs a single visualization-node equipped with a high-performance graphics adapter including modern functions and extensions. We develop an algorithm running on the

[†] This work is supported by the DFG-project DA155/31-1, ME872/11-1, FI1491/1-1 AVIPASIA
[‡] Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo)

DOI: 10.2312/EGPGV/EGPGV10/043-051

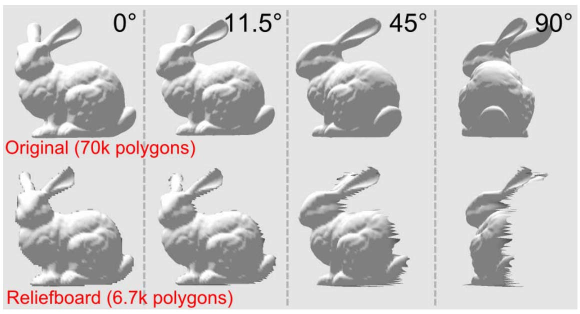

Figure 1: *Approximating objects by reproducing their surface by Reliefboards.*

visualization-node that uses the PC-cluster to compute only subroutines with simple graphics operations. Instead of rendering large parts of the scene frame by frame, the PC-cluster nodes assist the powerful visualization-node only. Therefore, these PCs do not even need graphic adapters. A software rendering system can be used as well. A requisite is that the sum of the different nodes' memory is large enough to store the scene redundantly. Modern features, e.g. new shaders can be used by the rendering algorithm running on the visualization-node.
Our rendering algorithm gains its acceleration by replacing and rendering the scene's distant objects with meshes consisting of a reduced number of polygons (*Reliefboards*). Reliefboards are computed during the walk-through of the scene (see Figure 1). The image quality of Reliefboards depends on the position of the viewer, i.e. the image quality becomes poor if the visitor moves away from the position where the Reliefboard was computed. The PC-cluster assists the visualization-node by computing simplified objects in regular intervals. We trade the number of cluster nodes off the scene objects' image quality. If we increase the number of cluster nodes, we increase the image quality. The rendering of both close objects as well as distant objects replaced by Reliefboards, is computed by a single visualization-node. Distant objects can be single objects given by the modeling of the modeler or groups of objects computed automatically by our clustering algorithm.

The rest of the paper is structured as follows: Section 2 gives an outline of the presented approach. Section 3 gives an overview on mesh replacement and parallel rendering methods and techniques. Section 4 describes how to cluster objects when the original objects of the scene can not be replaced with Reliefboards. Section 5 describes the creation of Reliefboards. Data distribution and load balancing are handled in Section 6. Section 7 shows the quality of the approximation and Section 8 provides a short summary and an outlook of future works.

2. Overview

Our system architecture consists of a visualization-node equipped with a powerful graphics adapter and a PC-cluster consisting of a large number of nodes equipped with weaker graphics adapters. The visualization-node renders the scene. We assume that a 3D scene consists of a certain number of elementary objects. The visualization-node renders parts of the objects with the original triangles. A large number of elementary objects are replaced by Reliefboards (see Figure 1). A Reliefboard is a mesh consisting of colored vertices placed on a regular grid and shifted along the grid's normal vector. Reliefboards approximate the original objects and their quality depends on the viewer's position. Therefore the Reliefboards are updated periodically. The PC-Cluster assists the visualization-node by computing the Reliefboards asynchronously.

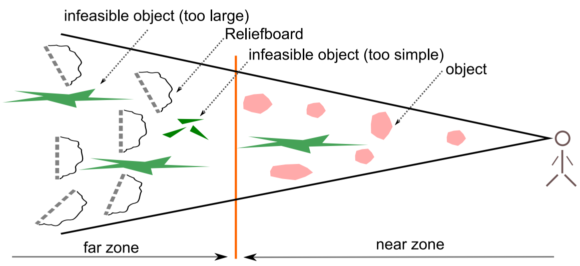

Figure 2: *Objects nearby the viewer, infeasible objects, and too simple objects are rendered with original polygons. Far away feasible objects are replaced and rendered with Reliefboards.*

Rendering cycle: A visualization-node's rendering cycle consists of three steps. First, all infeasible objects are rendered. We call objects, which are unsuitable for replacement *infeasible objects*. Infeasible objects are rendered with original geometry because their diameter is too large or the number of triangles is too small and thus a replacement with a Reliefboard would not gain any speedup (see Section 4). Second, all feasible objects are rendered in a front to back manner. An object is suitable for replacement with a Reliefboard if the number of triangles and the diameter of the object is adequate (see Section 4). We call the area where feasible objects are rendered with original triangles *near zone* (see Figure 2). Third, distant feasible objects behind the near zone are replaced and rendered with a Reliefboard. We call this area *far zone* (see Figure 2).

3. Related Work

We overview related work for mesh replacement, displacement mapping, and parallel rendering.

Mesh Replacement substitute objects' original geometry with a simplified representative mesh, which typically

reduces rendering costs (e.g., billboards or level of details (LOD)). Simple billboards replace original geometry by single textures placed on a rectangle. A disadvantage of this technique is that billboards can intersect at most once (see Figure 3).
Shade et al. [SLS*96] use textures to reduce scene complexity. They compute a kd-tree that contains the scene's geometry. By utilizing this kd-tree, the cell's geometry can be substituted by a rendered image of it, as seen from the observer's position. This technique's drawback is that sometimes it is possible to see through a gap between textures and geometry. Furthermore, if the position is changed there is no parallax effect. Reliefboards support the parallax effect, therefore the picture looks correct for a longer period.
Hoppe et al. presents in [HDD*93] techniques to optimize and simplify meshes, which can be used as LODs. For continuous LOD Hoppe presents *Progressive meshes* [Hop96]. These mesh simplifications must be generated in a preprocessing and the quality must be checked manually. Reliefboards are computed in parallel, automatically, and on-the-fly. They can be used for groups of different objects and generate a more simple representative for all of them at once.

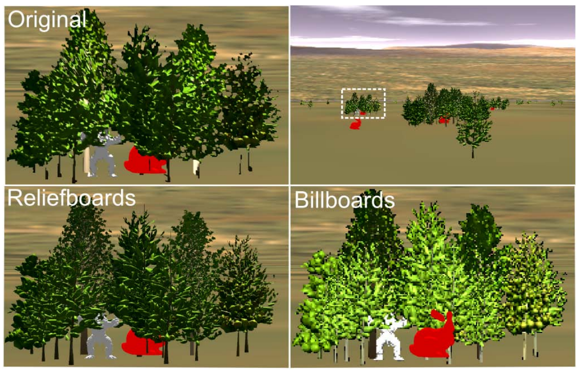

Figure 3: *The bunny is rendered with original geometry. The upper left picture shows a picture detail using Reliefboards. The lower left picture shows the original picture detail and the lower right shows the picture detail using simple billboards. These simple billboards neither support multiple intersections nor correct lighting.*

Displacement mapping is used to add depth to simple surfaces. In contrast to techniques like bump mapping, which only simulate the visual effects of depth (e.g. by lighting or shadowing), displacement mapping changes the geometry of a surface [Coo84]. These techniques support the parallax effect and intersection of objects. Typically, a simple surface is transformed to a more complex one by using several maps. By using these meshes correctly, shadowing and occlusion can be computed. Furthermore, objects' silhouettes appear realistic [WWT*03]. Often the underlying geometries are overtessellated, so that a retesselation is needed [GH99, DH00]. Typically, the displacement of the vertexes must be computed in every frame. If the number of vertexes to displace is high, this technique can influence the rendering time negatively. For this reason Reliefboards displace the vertexes not in every frame. The vertexes of the grid are shifted only one time for many frames.

Parallel rendering has been categorized into three basic approaches by Molnar et al. [MCEF94]: sort-first, sort-middle and sort-last. These approaches specify where and when geometric primitives are distributed in parallel running PC-cluster nodes. Samanta et al. [SFL01,SFLS00] presented several systems to combine these approaches. There are also several frameworks that provide parallel rendering in PC-clusters; some replace the OpenGL driver [HHN*02], others use shaders to exploit coherency between frames [AR05]. The main challenges are load balancing and PC-hardware's and network's latency. Response times are extremely important when frames are computed synchronously. Another problem is that the graphics adapters of the PC-cluster nodes have to support all needed features and extensions.
Our rendering approach distributes the different objects as in sort-last on the backend-nodes. Furthermore, the backend-nodes render their pictures independently from each other. Similar to *sort-last-sparse* every renderer computes only small parts of the frame and cuts out the needed values from the frame- and depth-buffer. But instead of sending these values to the visualization-node, the backend-nodes create meshes from this data. These generated meshes are send to the visualization-node, where they are used as approximation for multiple frames.
Although sort-last based approaches normally scale well in the number of render nodes in the network, the composition of the frames into one final image can easily result in a bottleneck. There are different hardware solutions to overcome the problems with the network or GPU-CPU communication [DY08,MOM*01]. But there are other disadvantages, like the limited number of applications that support such hardware. Rendering progresses have to be modified to use these accelerators. The *Lighting-2* module needs the depth buffer encoded in the framebuffer [SEP*01], for example. Another drawback is that this special hardware is inflexible and new techniques like PCI-Express allow even faster data transfer from GPU to CPU. Thus software solutions become feasible [EP08]. There are different techniques that can benefit from this, for example *Direct Send Compositing* or *Binary Swap* [lMPHK94]. But many older clusters do not support these techniques. In the *PC*^{2} for example, most of the PCI-Express lanes are dedicated to the Infiniband adapters.
Reliefboards combine the idea of billboards and automatically created displacement-maps. Reliefboards allow the application of a wide range of PC-clusters equipped without special graphics hardware.

4. Identifying Objects by Clustering

To apply Reliefboards, feasible objects are needed. The objects must have a high triangle count and must be compact in space. If the objects are given in a usable form, Reliefboards can be applied directly. Otherwise, if the objects are unfeasible or the geometry consists of triangles in arbitrary order, we segment the scene in several parts. Therefore, we use a heuristic with two phases, based on agglomerative clustering [DE84].

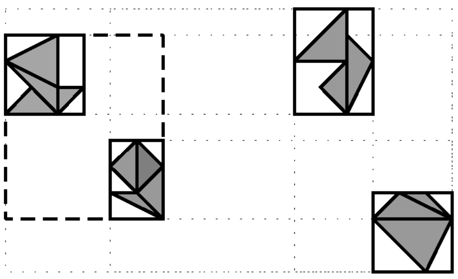

Figure 4: *Clustering based on the smallest joined bounding box*

In the first phase we determine the range of the near zone d_{nz}, where only original geometry should be rendered. Using this size we can calculate the side length of the clusters sl_c, that can be substituted by Reliefboards. For this calculations, we need the maximal amount of triangles t_{nz} rendered in the near zone, the Reliefboards' maximal projected side length in p pixels and the display resolution r as parameters. In the second phase we determine the required clusters with side length sl_c.

In the beginning of the first phase we remove redundant vertices and faces. Afterwards we determine connected components and their axis aligned bounding boxes (AABB) - that are the base-clusters for our agglomerative clustering algorithm. Until we calculated a cluster with t_{nz} triangles, the algorithm works as follows: Compute the joined AABB for all potential smallest cluster pairs. Union those two clusters, whose longest side of their combined AABB is minimal, with respect to all other cluster pairs. For optimization purposes, we do not compute all possible joined AABBs. Instead we compute only a constant number of AABBs for each cluster; we use only those clusters that are nearby. By this we reduce the clustering time from $O(n^3)$ to $O(n^2)$ where n is the number of connected components at the beginning. The cluster c_s that reaches a triangle count of t_{nz} firstly, is by definition the smallest we can reach with our clustering algorithm. The diameter of c_s is the range of our near zone d_{nz}. Using the determined d_{nz}, p and r, we can calculate the needed side length for the clusters sl_c.

In the second phase we use the base-clusters and sl_c in the agglomerative clustering algorithm. At first we remove all base-clusters whose maximal side length is longer than sl_c - these clusters should not be substituted by Reliefboards (that are infeasible objects). The algorithm enlarges the other clusters as long as their longest side is smaller than sl_c. When a cluster exceeds this limit it is completed and removed from further clustering. The resulting clusters are the objects, which are potentially replaced by Reliefboards. In Figure 5 these different types of objects are visualized.

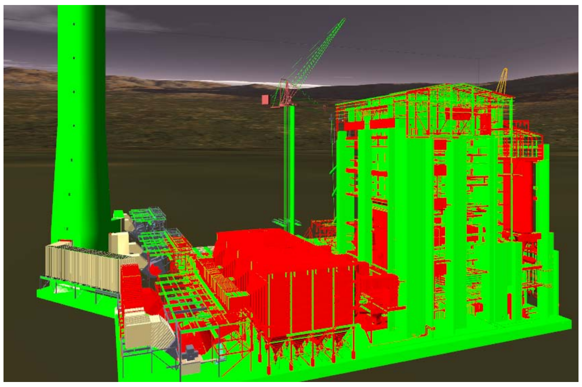

Figure 5: *Clustering of the power plant. Green (brighter) objects exceed the maximal side length of clusters and red (darker) ones are feasible clusters. Different colored parts are clusters, whose complexity is not large enough to replace them with Reliefboards.*

5. Reliefboard structure and creation

Reliefboard's foundation is a mesh consisting of colored vertices placed on a regular grid, which are shifted along the grid's normal vector. The general orientation of the grid's base plane is orthogonal to the vector from the center of the Reliefboard to the observer's position at the time when the Reliefboard is created. This absolute orientation stays fixed as long as the Reliefboard is valid. It does not change when the observer's position changes, unlike this may be the case for simple billboards that always face the observer.

The displacement of the vertices reproduces the surface of the approximated object as seen from original observer's position in orthographic projection – i.e. without perspective distortion. If the position of a vertex does not lie in the area that is covered by the original object's projection onto the Reliefboard's plane, this vertex is cut out from the mesh. As a result, the Reliefboard's silhouette reproduces the original object's silhouette (again, as seen from the observer's original position). The vertex-colors and vertex-normals are also taken from the projection of the unlighted original object. The combination of unlighted color and the normal vector allows exploiting the normal (possibly dynamic) lightning techniques, used during the Reliefboard's rendering on the visualization-node.

5.1. Creation

The data and parameters necessary for the creation of a Reliefboard are:

- The geometry (including colors, etc.) and the position of the original object,
- the directional vector from the observer's position to the original object's center (which also becomes the center of the Reliefboard)
- and the resolution of the grid. As the Reliefboard should reproduce the original object as good as possible at least at the time it is created, the resolution is set to the current projected size of the object in pixel. So every pixel is initially represented by one colored vertex.

For the computation of a Reliefboard, the original object's orthogonal projection needs to fit into an area on the given resolution's screen, the desired side facing towards the camera. Hence the object is initially translated, rotated and scaled accordingly. Then several intermediate two-dimensional color maps of the given resolution are created by rendering the original object onto the screen (or to an off-screen buffer). As we want to exploit a cluster's nodes with possibly not up-to-date graphics hardware for the generation of the Reliefboards, we do not make use of multi target rendering to render to multiple buffers at once. Instead, we use multiple rendering passes to obtain the data for the different target maps separately. Each of the four rendering passes exploits a simple shader to support the efficient processing of the data into the target map:

- The *distance-map* encodes the displacement by extracting the depth information of the pixels.
- The *stencil-map* marks which pixel are covered by the projection and which vertices should be cut out.
- The *color-map* reproduces the unlighted (but possibly textured) color of the object.
- And a *normal-map* encodes the surface normals, in order to support correct lighting.

Finally these maps are combined to form the mesh of the Reliefboard at the given position.

6. Distributed rendering

The parallel system used for rendering consists of two types of nodes: A group of backend-nodes is utilized for the distributed creation of the Reliefboards. One visualization-node requests and uses the Reliefboards to render the actual scene to the screen and provide the interface for the interactive movement of the observer.

6.1. Rendering on the visualization-node

The rendering of a frame on the visualization-node is performed in three steps. Initially all infeasible objects inside the viewing frustum which were identified as too large or to have a too low complexity are rendered to the screen. Their polygon count is thereby accumulated. In neither case would it be practical to replace these objects by Reliefboards. If an object is too large, even a slight change of the observer's position is likely to cause an unacceptable change in the perspective. If the object consists of too few polygons, the original object's rendering time could actually be better than the approximation's.
In the next step, the other objects are rendered in front-to-back order from the observer as long as the overall polygon count does not exceed the value of the parameter t_{nz}. If the number of objects is small enough, we can sort the objects using a standard sorting algorithm. Otherwise one should utilize a spatial data structure like a loose-octree for storing the objects and traverse it in front-to-back order.
This implicitly results in the subdivision of the frustum into near zone where only original geometry is rendered and the far zone where Reliefboards can be used (compare to Figure 2).
Finally, a decision whether the original object or the corresponding Reliefboard should be rendered, is made for each of the remaining objects in the viewing frustum. If no Reliefboard has yet been received for an object, the original object is rendered and – if no request for this object is pending – a new request is issued. Each request consists of an unique identifier for the object, the direction vector from the observer's position to the object's center and the object's projected size in pixel for the Reliefboard's resolution. Requests are processed asynchronously and the resulting data arrives after a few frames.

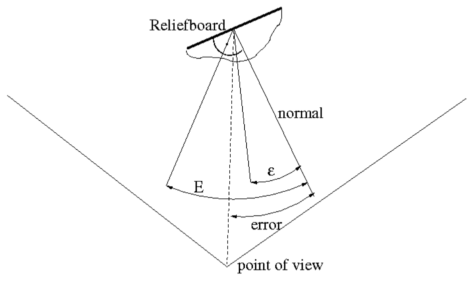

Figure 6: *When the angle-error exceeds ε a new Reliefboard is requested. If the angle-error exceeds E, the visual error is considered too large and the original geometry is rendered.*

If an Reliefboard has already been received, the current *angle-error* is calculated (see figure 6). The angle-error is defined as the angle between the direction vector that was used as parameter for creating the Reliefboard and the current direction vector from the observer to the center of the board. This is taken as an indicator for the visual similarity of the Reliefboard's projection and the original object. If the error is higher than a fixed parameter ε and no request for this object is pending, then a request for a new Reliefboard

incorporating the current direction is issued. If the error is unacceptably large (defined by another fixed parameter E), then again the original object is rendered until a new version is received. Another source for artifacts produced by a Reliefboard is the difference in the current projected size to the Reliefboard's original resolution. If the observer approaches a low resolution mesh, blocking artifacts occur. If the observer moves away from a high resolution mesh, aliasing artifacts occur. Hence we use another parameter indicating that relative difference in the projected size should also lead to an update request.
If a Reliefboard with acceptable error is available, the corresponding relief mesh is rendered at the original object's position, oriented towards the direction it was created in. This should be the most common case, as a Reliefboard can normally be used for many frames.

Also, in every frame it is checked whether newly created Reliefboards have been received from the backend-nodes. Then the corresponding old ones are replaced. This results in the occurrence of a slight popping effect in the next frame. This effect is more visible as the old and new direction vector differ. If the new Reliefboard replaces an original object, the Reliefboard's direction vector differs from the actual direction in which the object was seen last.

6.2. Distributed creation and load-balancing

One goal of the distributed creation of the Reliefboards is to obtain a latency for the requests as low as possible. When a request is delayed, it is possible that the out-dated old version of a Reliefboard is used for a longer period resulting in a increased visual error. If the request for an object, which is currently rendered originally, is delayed, even the rendering time may increase.
We initially distribute the data of all objects randomized and redundant, with a fixed number of replicas over the backend-nodes. In order to assign the requests to the nodes at runtime, we make use of a modified version of the c-Collision-Protocol [DadH93], which aims at having a low contention of every network node and spreading the requests evenly over the nodes (see Figure 7). This protocol is implemented on the visualization-node and is executed after each frame if new requests were made.
The distribution is performed in rounds. In every round a node handles at most c requests, but if it should answer more than c requests, it answers none. We begin with a value of c of 1. If not all requests can be handled with that value, c is increased until one further request can be fulfilled. Then c is reset to 1 and the distribution is continued until all requests are assigned to a node. The assignment of a request to a node persists, until the Reliefboard is received.
The dynamic handling of the value of c has the disadvantage of not being able to ensure a maximal contention as with the original protocol, but guaranties the handling of all requests, prevents the protocol from dead-locks and allows asynchronous communication.

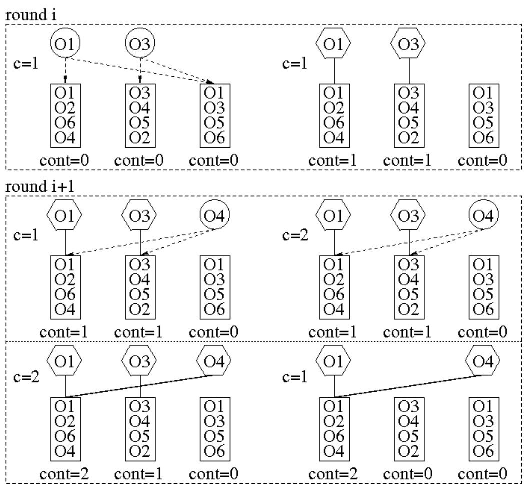

Figure 7: *Circles symbol requests and hexagons symbol handled requests. The data items are distributed redundantly and randomly among the different nodes*

7. Evaluation

In this section we present empirical results showing the practicability and the main characteristics of our methods.

7.1. Evaluation system

We implemented the presented techniques using the programming language C++ (GCC-3.4.0), OpenGL for rendering and MPI (ScaliMPI-3.12.0-1) for the communication between the nodes.
The PC-cluster we used for the evaluation is the Paderborn Center of Parallel Computing's (PC^2) Arminius cluster. The visualization-node provides a NVidia Geforce 9800GX2 graphics-adapter with 512 MiB memory (Nvidia OpenGL driver 1.90), PCI-e 1.0, two AMD Opteron 250 processors and 4 GiB RAM (The installed 64-bit OS is Fedora-Linux 9). The screen's resolution is set to 1024×768 pixels.
The 32 backend-nodes each provide a NVidia Quadro NVS280 graphics-adapter with 64 MiB memory (Nvidia OpenGL driver 1.77), PCI-e 1.0 (single lane), two Intel Xeon 3.2 GHz processors and 4 GiB RAM (The installed 64-bit OS is Red Hat Enterprise Linux AS release 4). The connection between all nodes is established via Infiniband adapters.

7.2. Scenes and parameters

We used two different test scenes: A forest composed of many (not instantiated) models of the Stanford bunny and different kinds of trees ($\approx$ 25 mio. triangles in overall 250 objects see Figure 3) and the well known power plant model

(≈ 12.4 mio. triangles). As the forest scene consists of relatively compact objects with almost uniform complexity only, it is possible to use these objects directly without any clustering. The power plant's original objects are very complex and lack locality (e.g. long pipes). Hence, to allow the proper application of Reliefboards, we defined 150 new objects by clustering the scene (see section 4) .
The camera path used for the power plant is shown in Figure 8. The wood scene is circled by the camera in 6000 steps.
In our first tests we started an update when an angle-error of $\varepsilon = 11.5^\circ$ was exceeded or when the projected side length of the Reliefboard differed more than 30% from the original. The original geometry was rendered when the angle-error exceeded $E = 45^\circ$. The value for t_{nz} is set to 3.5 mio. triangles.
The number of replicas for the objects placed on the backend-nodes was set to three, as a compromise between a sufficient distribution of the requests over the nodes and the increased demand for memory for additional copies.

7.3. Scalability

In the first test we evaluate the influence of the number of used backend-nodes on the request's average response time. This is the average time that passes from the initiation of a request until the corresponding Reliefboard has been received by the visualization-node. This includes the time for completing all prior requests on a node, rendering the original geometry, reading the framebuffer in the desired resolution several times (see section 5.1) and finally composing and sending the created Reliefboard to the visualization-node.
Our experiments show that an increased number of nodes decreases this response time, as the average congestion (the average size of the queue of open requests) on every node decreases (see Figure 9).
If many backend-nodes are available, the response time converges to the time needed for the construction of a single Reliefboard. So that in dependency of the scene and the amount of requests (implied by the speed of the observer's movement), an increased number of nodes does not pay off from some point.
The time needed for updating all Reliefboards in the scene (e.g. when the observer turns around very quickly) is not much higher than the average time needed for a single request, due to the parallel execution. For the power plant scene, the time for a complete update is nearly the same as the average response time; for the forest scene, the time is about twice the average response time.

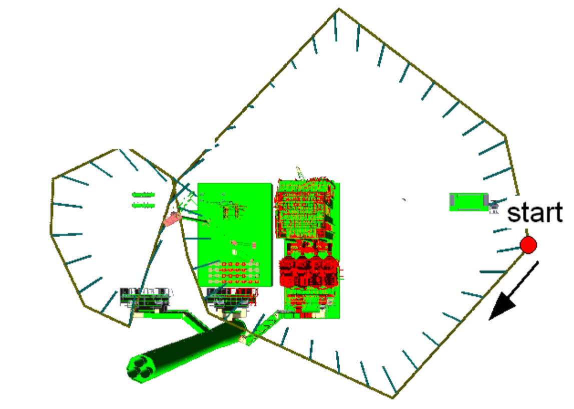

Figure 8: *Camera-path in the power plant scene. The path is traversed in 6000 steps at a uniform speed.*

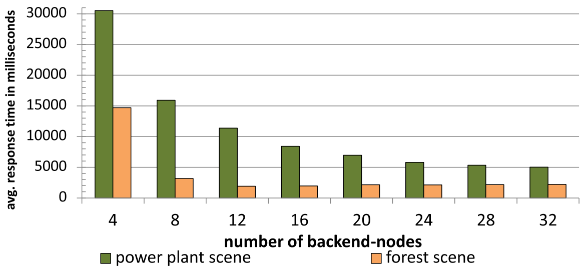

Figure 9: *Average response times for requested Reliefboards: The more nodes, the faster is the response – to the point where all requests can be processed immediately.*

7.4. Rendering time

One property of our method is, that the frame rate is mainly determined by the computational power of the visualization-node and the properties of the scene (i.e. the number of objects in the viewing frustum, the complexity of the unsuitable objects and the resolution of the Reliefboards). The number of backend-nodes does only slightly influence the rendering time, because of an overreaching latency, resulting in a high error, an original object is rendered instead of a Reliefboard. Figure 10 shows that the rendering time using Reliefboards can be reduced substantially if many objects are in the viewing frustum. The average rendering time along the chosen camera path using only frustum culling and the original objects is 43 ms, whereas the average rendering time with applied Reliefboards is only about 16 ms, nearly independent from the number of used backend nodes (the observed difference is below the error threshold of the measurements). Thus the frame rate for a given scene can be influenced by the hardware configuration of only a single PC, which can be improved by relatively cheap updates without modifying any other node of the cluster.

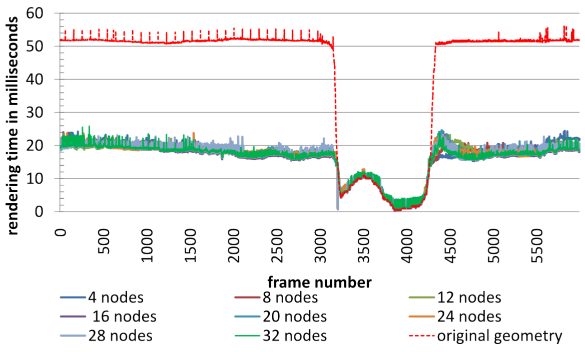

Figure 10: *Acceleration through Reliefboards in the power plant scene: The speedup is nearly independent from the number of nodes. The upper line shows the rendering time whithout Reliefboards.*

7.5. Scene sizes

The complexity of the scene for which the presented method is suitable is determined by several parameters:

The overall amount of available memory on the backend nodes needs to be big enough so that the scene can be stored redundantly with the chosen number of replicas. In our experimental setting, this was far from becoming a problem.

The computational power of the graphics adapter of the visualization-node mainly determines the achieved rendering time, which is composed of the time needed for rendering the original geometry and for the Reliefboards. If the number of objects in the scene gets too large, the frame rate may decrease because of too many Reliefboards that need to be rendered. Then the amount of rendered original geometry can be adjusted by lowering the parameter t_{nz}. This reduces the quality but leaves additional capacity for the rendering of Reliefboards.
In our prototypical implementation, the size of the scene is also limited by the amount of available memory of the visualization-nodes' graphics adapters. One should be able to lift this restriction by adding a simple memory management, which keeps the objects, that are located near the observer's position, and which are likely to get into the near zone within a few frame, in graphics memory.

Using another hardware setup consisting of nine desktop PCs (Visualization-node: Intel Quad Core 2.8 Ghz, 8 GiB RAM, NVidia 260GTX Hardware with 896 MiB graphics memory; various standard office PCs as backend nodes), we could render another forest scene with about 60 mio. triangles with 18 fps by applying Reliefboards (t_{nz} = 3.5 mio, 550 objects displayed as Reliefboard). When solely rendering the original scene on the visualization-node we only achieve about 2 fps.

7.6. Visual quality

Although the frame rate is nearly independent of the number of backend-nodes, the visual quality is heavily influenced by the response time and therefore by the number of nodes. The less the response time is, the less is the visual error, which can be caused by an outdated Reliefboard. Figure 11 shows the average distribution of the angle-error for the power plant scene. The difference in the projected size is not considered as the influence is insignificant due to the chosen camera path. As expected, more nodes decrease the occurrence of errors. For the chosen setting, not more than 28 nodes seem to be sufficient to achieve a mean error of only 16 degree (compare to Figure 1). Additional nodes are not necessary as almost each request can be processed immediately.

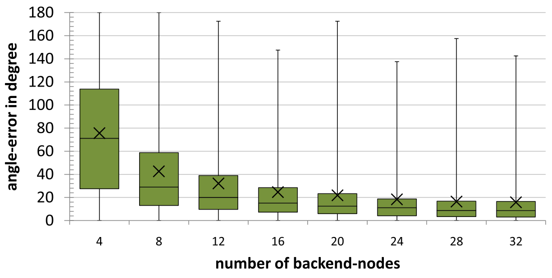

Figure 11: *Accumulated angle error in the power plant scene (with min, max, quartiles and avg values): The visual quality scales with the number of available nodes.*

8. Conclusions and future work

In this paper we presented a method that allows to use wide range of PC-clusters with only limited graphic capabilities for parallel rendering, so that only one powerful visualization-node is needed. The techniques accelerate the rendering process by utilizing weaker backend-nodes to generate approximations of parts of the scene on-the-fly. The presented approach uses an asynchronous communication scheme and distributes the load to the different rendering-nodes mostly evenly. If the number of render-nodes is increased, this rendering method scales in the quality of the rendered picture instead in rendering speed.
In the future, we plan to combine Reliefboards with other techniques to further reduce the popping effects on updated Reliefboards (e.g. by using blending [MH99]). The presented PC-cluster configuration consists of two different types of nodes only. We also plan to extend the protocol used for the distribution of the requests to support backend-nodes with different graphic capabilities.

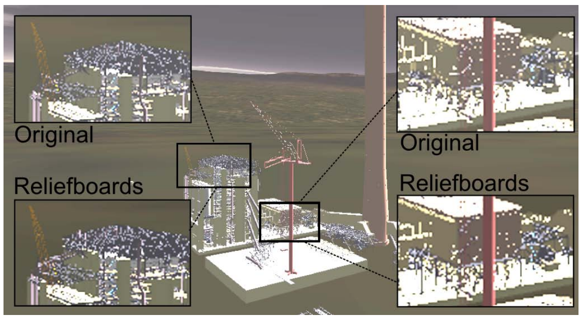

Figure 12: *The upper detail pictures show a part of the power plant rendered with original geometry. In the lower detail pictures Reliefboards are utilized.*

References

[AR05] ALLARD J., RAFFIN B.: A shader-based parallel rendering framework. In *IEEE Visualization Conference* (Minneapolis, USA, October 2005). 3

[Coo84] COOK R. L.: Shade trees. *SIGGRAPH Comput. Graph. 18*, 3 (1984), 223–231. 3

[DadH93] DIETZFELBINGER M., AUF DER HEIDE F. M.: Simple, efficient shared memory simulations. In *SPAA '93: Proc. Symposium on Parallel algorithms and architectures* (New York, NY, USA, 1993), ACM, pp. 110–119. 6

[DE84] DAY W., EDELSBRUNNER H.: Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification 1*, 1 (December 1984), 7–24. 4

[DH00] DOGGETT M., HIRCHE J.: Adaptive view dependent tessellation of displacement maps. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (New York, NY, USA, 2000), ACM, pp. 59–66. 3

[DY08] DOMINICK S., YANG R.: Anywhere pixel router. In *PROCAMS '08: Proceedings of the 5th ACM/IEEE International Workshop on Projector camera systems* (New York, NY, USA, 2008), ACM, pp. 1–2. 3

[EP08] EILEMANN S., PAJAROLA R.: Direct send compositing for parallel sort-last rendering. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses* (New York, NY, USA, 2008), ACM, pp. 1–8. 3

[GH99] GUMHOLD S., HÜTTNER T.: Multiresolution rendering with displacement mapping. In *HWWS '99: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (New York, NY, USA, 1999), ACM, pp. 55–66. 3

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. In *SIGGRAPH '93: Proc. Conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), ACM, pp. 19–26. 3

[HHN*02] HUMPHREYS G., HOUSTON M., NG Y., FRANK R., AHERN S., KIRCHNER P., KLOSOWSKI J.: Chromium: A stream processing framework for interactive graphics on clusters, 2002. 3

[Hop96] HOPPE H.: Progressive meshes. In *SIGGRAPH '96: Proc. Conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 99–108. 3

[lMPHK94] LIU MA K., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap image composition. *IEEE Computer Graphics and Applications 14* (1994), 59–68. 3

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: *A Sorting Classification of Parallel Rendering*. Tech. Rep. TR94-023, 8, 1994. 1, 3

[MH99] MÖLLER T., HAINES E.: *Real-time rendering*. A. K. Peters, Ltd., Natick, MA, USA, 1999. 1, 8

[MOM*01] MURAKI S., OGATA M., MA K.-L., KOSHIZUKA K., KAJIHARA K., LIU X., NAGANO Y., SHIMOKAWA K.: Next-generation visual supercomputing using pc clusters with volume graphics hardware devices. In *Supercomputing '01: Proc. Conference on Supercomputing* (New York, NY, USA, 2001), ACM, pp. 51–51. 3

[SEP*01] STOLL G., ELDRIDGE M., PATTERSON D., WEBB A., BERMAN S., LEVY R., CAYWOOD C., TAVEIRA M., HUNT S., HANRAHAN P.: Lightning-2: A high-performance display subsystem for PC clusters. In *SIGGRAPH 2001, Proc. Conference on Computer graphics and interactive techniques* (2001), pp. 141–148. 3

[SFL01] SAMANTA R., FUNKHOUSER T., LI K.: Parallel rendering with k-way replication, 2001. 1, 3

[SFLS00] SAMANTA R., FUNKHOUSER T., LI K., SINGH J.: Hybrid sortfirst and sort-last parallel rendering with a cluster of pcs, 2000. 1, 3

[SLS*96] SHADE J., LISCHINSKI D., SALESIN D. H., DEROSE T., SNYDER J.: Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH '96: Proc. Conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 75–82. 3

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: View-dependent displacement mapping. *ACM Trans. Graph. 22*, 3 (2003), 334–339. 3