Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Sorted Pipeline Image Composition

Marcus Roth[†1] and Dirk Reiners[‡2]

[1]Fraunhofer IGD, Darmstadt, Germany
[2]Virtual Reality Applications Center, Iowa State University

Abstract

The core advantage of sort last rendering is the theoretical nearly linear scalability in the number of rendering nodes, which makes it very attractive for very large polygonal and volumetric models. The disadvantage of sort last rendering is that a final image composition step is necessary in which a huge amount of data has to be transferred between the rendering nodes. Even with gigabit or faster networks the image composition introduces an overhead that makes it impractical to use sort last parallel rendering for interactive applications on large clusters. This paper describes the Sorted Pipeline Composition algorithm that reduces the amount of data that needs to be transferred by an order of magnitude and results in a frame rate that is at least twice as high as the widely used binary swap image composition algorithm.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics SystemsDistributed/networked graphics;

1. Introduction

High-performance computing nowadays is dominated by clusters of simple systems. The economics of scale make it more cost-efficient to use lots of cheap components, even if they are not quite as well integrated and have less performance than a large, single unit. The same is true for graphics systems, where affordable PC-based systems dominate the market. The parallel development in cheap projectors has given rise to a large number of tiled screen system for high-quality, high-resolution displays, nearly always driven by a cluster of Common Off The Shelf PCs. Using cluster-capable software [Rot02, HHN*02, Ols02] it is easy to distribute the rendering tasks over multiple machines to generate the different parts of the large display. However, using the combined rendering power of a whole cluster to drive a high complexity model to a single display is done much more rarely.

The main reasons are the large bandwidth demands and higher software complexity this places on the whole system. It is relatively straightforward if the screen can be broken up into small pieces, each of which is rendered by a single node (sort-first [Mol91]). In this case, every node just renders a smaller viewing frustum and transfers its screen part to the display node, which adds up to a single screen worth of data.

The problem with this approach is its scalability. With a growing number of nodes it moves significantly below the desired linear speedup. This is caused by the overhead of rendering the pieces of the scene that straddle tile boundaries on every affected tile [Mol91]. With a large number of nodes, this overhead alleviates all the gains of adding nodes to the system.

This overhead can be avoided by splitting the scene instead of the screen into independent parts, and assigning scene parts exclusively to individual nodes (sort-last). In comparison to sort-first this approach makes good use of the available rendering performance, scales linearly and has the additional benefit that it is possible to work with scenes that are significantly larger than the memory of a single node. But the integration of the partial results becomes more costly, as pixels used by multiple objects need to be transferred from all the affected nodes, usually with depth information for occlusion resolution. In the special case of volume rendering this depth transfer can be avoided, as volumes can easily be split into non-overlapping parts. This makes sort-last meth-

† e-mail:mroth@igd.fhg.de
‡ e-mail:dreiners@iastate.edu

ods especially attractive and many of the previous methods have been developed in this context. For polygonal scenes the splitting is possible, but it is much more costly and introduces similar scalability constraints as sort-first rendering.

A number of different approaches have been developed to attack the data volume problem for sort-last composition, but the most effective ones have some undesirable limitations.

2. Previous Work

In the following the most commonly used approaches for sort-last image composition are discussed.

Central composition [NZ00]: Central composition is the most simple image composition strategy. Each parallel rendering node sends its image with depth information to a single display node, which composes all images according to the pixel depth values to a final image. As a consequence the amount of data grows linearly with the number of nodes, all of which needs to be processed by the display node. It is simple to implement but scales only for a very small number of parallel rendering nodes.

Binary Composition [Hei94, RS99]: The binary image composition tries to relieve the burden on the display node by parallelizing the composition step. In each step pairs of nodes combine their respective partial results with one node sending its image to the other, and recursively combining the intermediate results until the final image is created. The main disadvantage of binary composition is that is does not fully use the available computing power of the cluster, as in the second step already only half of the nodes are involved, with successively smaller percentages in subsequent steps. Even if the binary composition is much more efficient than the central composition, its resource usage is far from optimal.

Direct Send [MPHK94, MWP01]: In direct send image composition each parallel renderer is responsible for a separate region of the final image. After each node has rendered its local image, it splits up the image and sends each part to the responsible host. In contrast to the binary composition all nodes are working during the whole composition and the display node only needs to receive the final image (similar to sort-first), but the overall amount of data is not reduced, which limits scalability.

Binary Swap [MPHK94]: The core idea of binary swap is to alleviate the load imbalance of binary composition. The difference is that during each composition step two nodes swap only *half* of their image with each other. After a composition step each node has half of the composed image of two nodes. In the next step new nodes pairs are created. Two nodes swap a quarter of the remaining image with each other. At the end of the composition each node holds a part of the final image. The amount of data that has to be transferred is equal to direct send. Two effects work together to reduce the amount of data that is transfered in binary swap. In the first stages every node only has the data it rendered itself to transfer, which for optimized models is rather compact, allowing a sparse approach (i.e. only transfering the parts of the image that are actually covered by objects) to transfer only a small part of the full screen. In later stages only small parts of the full image are under consideration, also keeping the amount of data to be transfered for each node low.

Pipeline Composition: Pipelines are mostly used in hardware image composition. Examples are PixelFlow [MEP92], Orad's DVG [Ora04] or Sepia [MHS99, HM99]. All or part of the image is sent through a pipeline. In each pipeline stage one node composes the incoming pixels with its local pixels and sends the result to the next pipeline stage. The final composed image is then located at the last node in the pipeline. As composition is done on each pixel independently it is possible to forward processed pixels to the next stage in the pipeline as soon as the processing is done. This can keep the latency per pipeline stage very low. One differentiator between different systems is whether the pipeline order is defined by the actual physical connection of nodes or whether it can be reconfigured in software, and the different composition modes available.

Parallel Pipeline Composition: In [LRN96] Lee, Raghavendra and Nicholas describe a modified pipeline composition, where the nodes are connected to each other in a circle. Each node is responsible for one part of the screen, and sends all regions for which it is not responsible to the next node in the ring. After n-1 steps each node holds a composed image for the region for which it is responsible, which then needs to be transfered to the dispay node. The parallel approach avoids the initial phase of filling the pipeline, which can be rather long, especially for software-based pipelines.

Hybrid Sort-First and Sort-Last Parallel Rendering: The amount of data that has to be transferred for image space parallel rendering (sort-first) is much smaller than that of sort-last. [SFLS00] tries to unite the efficiency of sort-first and the scalability of sort-last by combining them. It tries to split the scene in a way that allows parts of the screen to be rendered by a single node. For these parts, no depth compositing is necessary, eliminating a large amount of network traffic. The remaining parts are distributed among the nodes and composed using binary swap. Therefore the worst case is a standard binary swap, while the best case is equal to sort-first composition. This approach can yield very high efficiency, but only under certain circumstances.

The efficiency depends strongly on the locality of objects in the scene. If large objects cover the whole screen, it is not possible to get a good load balancing and also find exclusive regions on the screen. Thus to get good efficiency the scene needs to be split up into a large number of small parts, to increase the probability of finding exclusive parts. This splitting might not be possible for some applications that depend on the integrity of the objects in the scene. Even if the

split is possible, it comes at a cost. Larger numbers of objects result in a more expensive load balancing step, as the screen footprint of each object needs to be calculated to detect exclusivity and to assign scene parts to nodes. This cost reduces the benefit over sort-last, which needs no load balancing calculations, but also performs well on scenes with many small objects. But the hybrid approach shares one of the main drawbacks of sort-first rendering, in that every node needs to be able to render any part of the scene in order to have a chance for exclusivity. As a consequence the maximum model size is limited to that of a single machine and not the potentially much larger combined capacity of the cluster.

Image Layer Decomposition: Another approach to reduce the amount of data was proposed by Nguyen and Zahorjan [NZ00]. The scene is divided in parts that can be sorted back to front. These parts are then composed by layering without the need to transfer the depth component of a pixel. For dynamic scenes where objects are moving the subdivision is view dependent and has to be done per frame. The disadvantages for large scenes are the same as for the previous hybrid approach.

Sort-last approaches have very desirable advantages over sort-first and hybrid approaches (scalability, ability to handle large scenes), but they come at a high price in terms of network load.

In addition to the number of pixels the performance of the image composition is dependent on the number of messages that have to be transferred. This is relevant because most cluster interconnects introduce a constant overhead for each message independent of the size of the message. Pipeline methods are very efficient here, but they need elaborate hardware implementations to overcome latency constraints. Binary swap is the most efficient pure software-based approach in this respect, it is the most widely used and therefore serves as a benchmark to compare against.

3. Contributions

The main contributions of the Sorted Pipeline method presented in this paper are

- Cross-node occlusion culling is employed for bandwidth reduction.
- Significantly reduced network bandwith compared to algorithms like Binary Swap, typically a factor of 4 less per individual nodes, and a factor around 10 when looking at the overall network bandwidth.
- As a pure sort-last approach it allows handling models as big as the sum of all nodes' memory. No model transfer for load balancing is necessary.
- No visibility or geometry preprocessing necessary, fully dynamic scenes are handled directly.

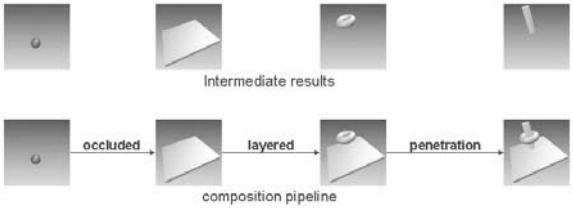

Figure 1: *Depth relation situations*

- Arbitrary numbers of nodes are supported, not limited to powers of two.

4. Sorted Pipeline Composition

The basis for the Sorted Pipeline Composition is the pipeline approach. Each node reads pixels from its predecessor. If a read pixel is in front of a local pixel, then the value is passed unchanged to the following node. Otherwise the local pixel is sent to the successor in the pipeline. After all pixels are processed, the final image is available at the node at the end of the pipeline.

To improve on the original algorithm global information is used to avoid unnecessary data transfers and to reduce the length of the pipeline, in addition to splitting up the image into tiles to allow parallelism to improve usage of the cluster nodes.

Assuming there is a global knowledge of depth values, then data transfers can be avoided in the following situations (see fig. 1):

a) If all pixels are filled with the background, then the current node can be removed from the pipeline.

b) If on one node all pixels are set and no background is visible, then this node *occludes* all other nodes that have no pixels that are in front of the farthest pixel of its image. All occluded nodes can be ignored by the image composition and can be removed from the pipeline.

c) If all pixels of the current node are behind all pixels of following nodes, then the pixels can be combined simply by *layering* the current node behind the following node. A depth value comparison is not necessary.

Situation d) is the default case with *penetrating* depth ranges that need depth composition.

The data reduction possible with these rules depends strongly on two factors. The first is the ordering of objects in the pipeline. If they are sorted back to front rules b) and c) will apply more often, significantly reducing the need to transfer data. The second is the granularity of the test, between the two extreme approaches of a per-pixel test and a global per-image test. The following section describes the approach used to optimize both factors.

4.1. Overview

In contrast to the original pipeline composition the screen is split into equal sized tiles. For each tile an independent pipeline composition is done, giving each area its own pipeline. As described above there are some situations where a simplified composition is possible. To raise the likelihood of such situations each pipeline is sorted in a way that areas with pixels that are far away from the viewer are first in the pipeline and areas with pixels near to the viewer are at the end of the pipeline. The farthest pixel of each area that is not a background pixel is used as a sorting key to achieve this. Nodes whose tiles are empty or fully occluded are removed from the pipeline.

With this approach a huge part of the normally required data transfer can be avoided. In a cluster environment where the composition is mostly limited by the network bandwidth, this data reduction leads to higher frame rates, or the performance improvement can be used to render more complex objects.

The main problem in realizing this sorted pipeline composition is the global data gathering and the independent sorting of many composition pipelines. An additional optimization factor is choosing the size of the composition area to get the best performance.

4.2. Pipeline Sorting

As described before data reduction can be increased if the visible objects in a pipeline are sorted from back to front. For this sorting it is enough to take the pixel farthest from the viewer as the sorting key. There are two possible solutions to form a sorted pipeline.

It would be possible to assign each node a fixed position in the pipeline and then assign each node a part of the scene in such a way that the first node renders the farthest and the last node renders the nearest part of the scene. This approach has the disadvantage that the assignment of scene parts is view dependent and as a result of this each node must be able to render all parts of the scene. Additionally this kind of sorting only provides a global sorting for the whole image. For the small image area composition described below, a local sorting will be more efficient.

Using a switched network like Ethernet allows dynamic reordering of the pipeline, and therefore assigning scene parts exclusively to any given node (pure sort-last) is possible. As a consequence the distance of the closest and farthest pixel of a node are view-dependent. They are calculated by each node for each frame and are transferred to a master node, which sorts the nodes in the pipeline back to front.

4.3. Gathering A Global State

To be able to do pipeline sorting and optimization, information of all parallel rendered images has to be transferred to the master node. For each node the depth of the nearest and the farthest pixel for each tile are needed. Additionally for the occlusion test it is necessary to know if any background pixels are visible.

To find the nearest and farthest pixels the depth buffer needs to be read back into main memory. An approximated test would be possible based on objects' bounding volumes, but that would significantly limit the possible gains.

The time cost for this operation differes widely from card to card, for the nVidia QuadroFX 1100 used in this paper about 50 million depth buffer values can be read back per second. Color frame buffer readbacks need about the same time, which allows reading back a 1280x1024 screen 19 times per second. In practice less time is needed, as for most nodes the visible geometry does not cover the whole screen, but buffer readback accounts for a major part of the time used.

For the pipeline check it is necessary to find out whether a part of the screen is totally empty or if it occludes the background completely. This can be done in software or using the GL_ARB_occlusion_query OpenGL extension by drawing a rectangle at the far clipping distance. The number of visible pixels for this rectangle defines occlusion or visiblity. Our implementation shows that in many cases a speedup is possible, but unfortunately composition is time consuming if the occlusion test fails. In these cases the time for the occlusion test increases the overall composition time. Therefore the occlusion test makes fast situations faster and slow situations slower. The presented implementation does not use the occlusion test to optimize for the worst case.

After the min and max depth and occlusion values for all tiles and all nodes are calculated, they have to be transferred to a central node which is then responsible for the optimization and pipeline definition. This data gathering process is a potential bottleneck.

The amount of data depends on the number and size of the composition tiles. More tiles increase the chances for hitting one of the efficient cases listed above, but they also consume more collection bandwidth. A simple optimization is using run-length encoding for the transfer. The geometry of a single node will only touch a small number of the possible tiles, run-length encoding will compress the data for the unused tiles very efficiently, on average by a factor of 10. But the tile size still remains an important optimization factor.

The extreme cases are separate pipelines for each pixel and a single pipeline for the whole screen. Fig. 2 shows the relation between the tile size and the total amount of information that needs to be transfered, using the Lucy animation path as an example. For small tile sizes the data transfer for the tile sort dominates the data transfer whereas the transfer for the

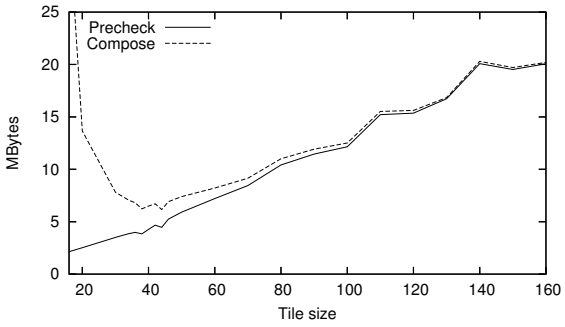

Figure 2: *Relation between tile size and transferred data*

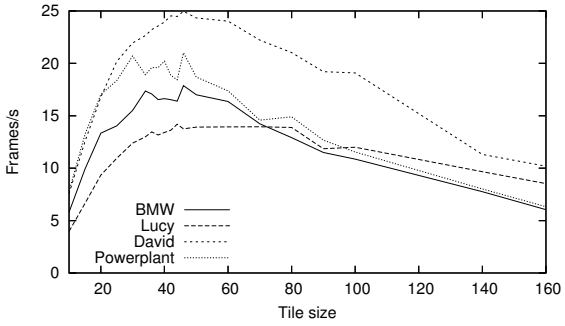

Figure 3: *Relation between tile size and framerate*

composition is very small. For very large tiles transfer for the sorting is small but data reduction is also small so a lot of data has to be transferred for the image composition. A minimum of the overall data load can be found for 40-50 pixel tiles, with rather similar values for the different scenarios. Fig. 3 shows the rendering performance with different tile sizes. For the rest of this work 40 pixel tiles were used.

4.4. Optimizing The Pipeline

Not all parallel rendering nodes contribute pixel values to all regions of the display. Therefore the pipeline length can be reduced by removing all nodes that contribute no pixel or only background pixels to a region. Further reduction is possible by removing all nodes where all pixels are occluded by another node.

An area is completely occluded if on one node no background pixel is visible and no depth value of the area is smaller then the farthest depth of the occluding node. This is the most efficient optimization as it allows drastic shortening of the pipeline. It occurs in all scenes where parts of the scene are occluded by walls or other features. In the used tests it occurred often in the BMW model where the carriage covers a lot of the internal geometry. It rarely occurs in the scanned models like the Stanford Lucy because the model has very few self-occluding parts and low depth complexity.

The tiles that are not occluded and not empty are then sorted from back to front, using the the maximum depth value as the key. Looking at the smallest depth value that can come out of the previous node in the pipeline allows deciding whether depth composition is necessary or not. If it is not needed, the depth buffer transfer can be skipped and simple layering can be used instead for the composition.

As a result of all optimizations we have a position in a pipeline for each tile, and whether depth composition is necessary or not. This information is then transferred to each node. Similar to the data compression for data gathering a simple run length encoding is used to efficiently compress information about empty or occluded tiles.

Fig. 4 shows the pipeline length for three models, distributed across 32 nodes, resulting in an initial pipeline length of 32 for all regions. In the power plant model a lot of geometry is located in the building, which creates a rather long remaining pipeline. Even in this case the longest remaining pipeline has length 23, less than three quarters of the initial length. The average pipeline length for the whole image is 1.37. For the BMW model the maximum length is shorter (16), but the complex regions cover a larger part of the screen, resulting in an average pipeline length of 1.628. In the Lucy model the small covered screen space gives rise to a lot of empty areas that push the average down to 1.01, and because of the relatively low depth complexity the maximum pipeline length is limited to 10.

4.5. Image Composition

After each rendering node gets its pipeline information, the composition is started.

As each tile's pipelines are totally indepepdent of each other, they can all be processed independently and in parallel. The process is started from the tiles that have no incoming pixels, these are just sent out to the next node in the pipeline by a sender thread. An independent reader thread waits for incoming data and puts it into a processing queue, which composes the data and puts into the the queue of the sender thread.

This data-driven on-demand approach ensures maximum utilization of the available bandwith.

5. Results

5.1. Test Scenarios

The described approach has been implemented in the OpenSG Open Source scenegraph and is available as part of the OpenSG system. All tests were run on a 48 node cluster running 2.4 GHz Pentium 4 machines with nVidia QuadroFX 1100 graphics cards, generating a final output

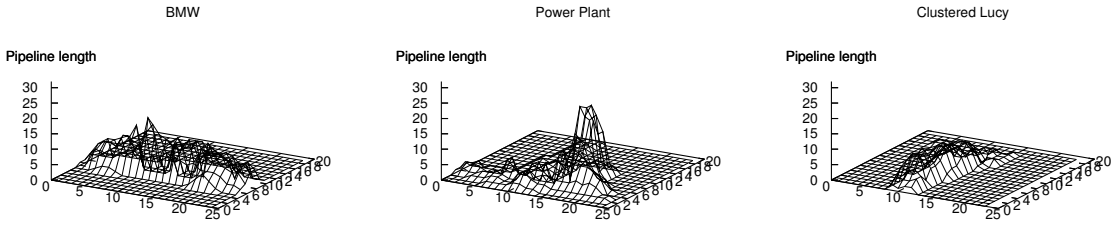

Figure 4: *Tile pipeline lengths for the different models*

image of 1024x768 pixel. The computers are directly connected to a Cisco gigabit switch.

The described algorithm was tested on different types of scenes. The first is the well known power plant from the University of North Carolina. It consists of 13 million polygons, and is primarily interesting because of its significant depth complexity, which makes it a very interesting case for the pipeline optimizations described above. The second is a model of a BMW 7 series car, with 3 million polygons. It has a widely varying object sizes (e.g. the whole carriage is a single object, while the interior is finely split), but good occlusion. The third scene type are high-polygon scanned models from the Stanford 3D Scanning Repository. Lucy (28 million polygons) and David (56 million polygons) were used. Lucy was used in two different versions. One was created by striping the full model and selecting random strips to split the model into pieces, the other using a clustering-based algorithm similar to [IG03]. The first version has object pieces that span a large part of the model (long strips) and consequently cover large parts of the screen, the pieces of the second are much more compact. For all models an animation path was generated that includes close-up and overview shots.

All of these models don't really tax todays graphics cards any more, which can be seen in the small amount of time taken by the actual rendering in fig. 8, but they represent different model characteristics that are relevant to the performance of the presented algorithm, emphasizing the importance of the transfer bandwidth.

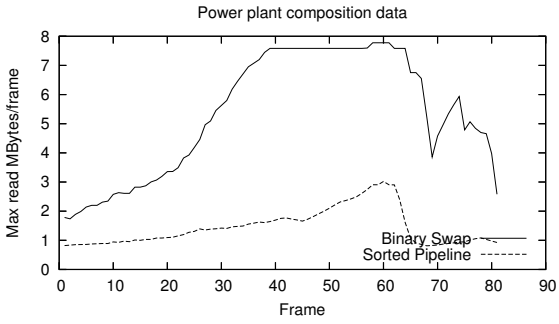

Figure 5: *Maximum data received by a single node per frame*

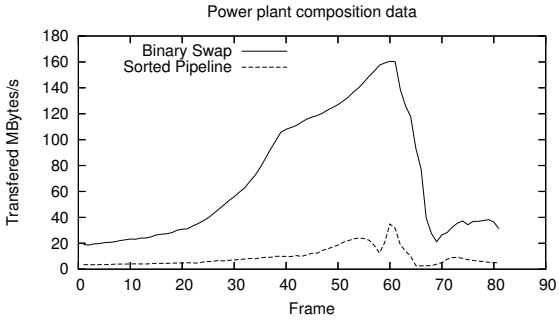

Figure 6: *Sum of all transferred data for all nodes*

5.2. Bandwidth Reduction

To judge the quality of the algorithm independent of the specific implementation the amount of data that needs to be transfered per frame was recorded for the power plant animation path. To put the results in perspective the binary swap approach was also implemented and used as a comparison. Both algorithms are optimized and use a sort-last-sparse approach, which is partially responsible for the large variability in the frame rate.

Fig. 5 shows the maximum amount of data that needs to be transfered into a single node, which can then quickly become the bottleneck for higher resolution images. The binary swap approach needs to transfer between 2 and 8 times as much data into a node, with an average around 4.

The situations gets even worse when looking at the total amount of data that needs to be transfered (see fig. 6). This can become a serious problem for a switched network, as most switches have a lower internal bandwidth than the sum of all connections, i.e. not every node can send to another node at full speed at the same time. The Cisco gigabit switch used on the test cluster for example provides an internal

bandwidth of 6 gigabit/sec. In this metric binary swap on average needs to transfer between 4 and 34 times as much data as that sorted pipeline, with an average around 10.

5.3. Performance Improvement

Figure 7 shows the impact on actual framerate that is incurred by the overhead on the data side. The power plant is very dense in the core of the model, leading to very long pipelines and limited gain. Nonetheless, the sorted pipeline is always faster that binary swap. On average 65% faster, in the best case 3 times faster. The BMW model is a good example for the sorted pipeline, as there is a lot of occlusion to be exploited. It renders 3-4 times faster than binary swap. The Lucy model with its low depth complexity benefits less, but still significantly. On average it renders 2 times faster than binary swap.

Figure 8 shows the time distribution for the different tasks. The two most prominent parts are reading the frame buffer and the composition. The pipeline sorting and the actual rendering only take up a small part of the time, especially in a close-up view the frame buffer read clearly dominates the needed time.

This large impact of buffer reads has both good and bad aspects. It is bad because there is very little that can be done about it, it just takes a certain amount of time to read the frame and depth buffers. The good part is that it is on the development curve of graphics hardware. Using PCI Express connections and newer graphics cards, the readback time will be reduced significantly. Another good aspect is that it can be done in parallel with composing the previous image. This would allow a further improvement in frame rate of a factor of up to 2, at the cost of not reducing latency. This is left as future work.

All tests are showing that a noticeable speedup is possible with the sorted pipeline algorithm. Excluding the time for rendering and buffer read which are equal for all sort-last algorithms, the composition with the sorted pipeline algorithm is at least twice as fast as the binary swap.

5.4. Scalability

Figure 9 shows the speedup for rendering with different numbers of parallel rendering nodes, averaged over the whole animation path. It shows that an arbitrary number of nodes can be used with the sorted pipeline, compared to binary swap which only works with power-of-two cluster sizes. It also shows the impact of using clustered vs. unclustered model splitting, which allows the sorted pipeline to run much more efficiently as it increases the probability of occlusion events. This is caused by the compactness of the clustered model parts and the fact that they have fewer holes, covering more area completely.

It shows that it's easily possible to handle dynamic billion polygon models without any preprocessing or simplification with parallel polygon rendering.

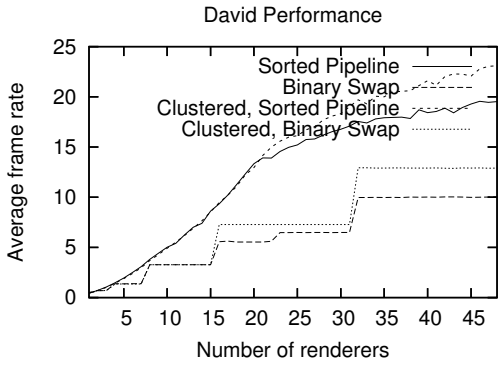

Figure 9: *Scalability compared to Binary Swap, for clustered and unclustered models*

6. Future Work

The core ideas presented here are orthogonal to some of the hybrid sort-first/sort-last ideas presented in [SFLS00]. It would be possible to combine them to further reduce the amount of bandwidth needed by replacing their binary swap with a sorted pipeline.

As the two most expensive parts (readback and composition) are independent, it would be possible to do them in a pipelined fashion to increase performance. The OpenSG clustering framework is not yet designed for that, but it's a logical step.

Acknowledgements

Parts of the described research were funded the German Ministry for Research and Education (BMBF) in the OpenSG Plus project.

We would like to thank the University of North Carolina and their anonymous donor for releasing the power plant model, and the Stanford Digital Michelangelo Project for the Lucy and David models.

We would also like to thank the authors of [IG03] for actually citing the libraries that they used (Metis and ANN), as that enabled us to develop the out-of-core clustering tool in a single day. Information sharing about implementation basics is a good thing!

References

[Hei94] HEILAND R.: Object-oriented parallel polygon rendering. In *ACM Graphics and Visualization Conference* (1994), ACM, pp. 19–26. 2

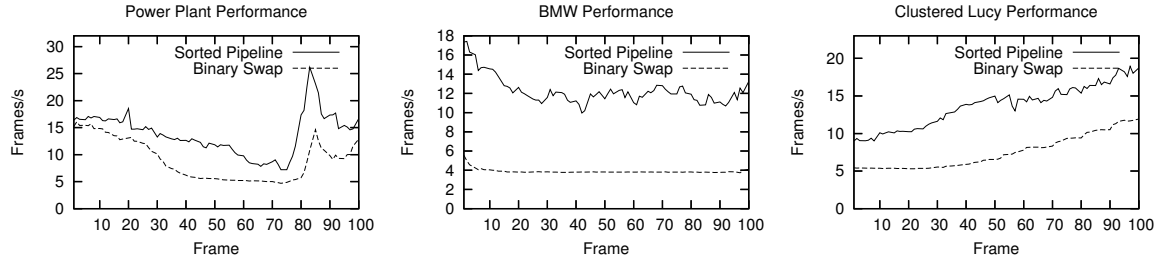

Figure 7: *Performance comparison between sorted pipeline and binary swap*

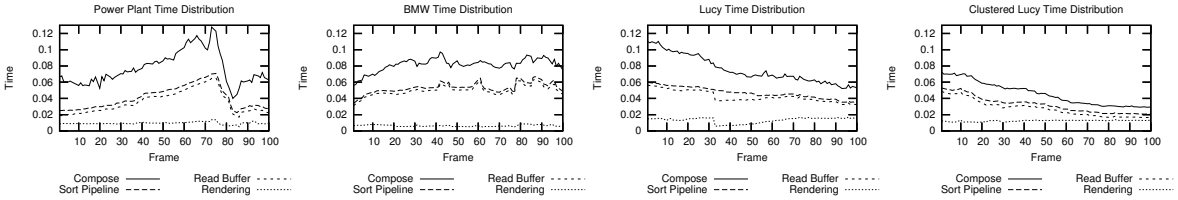

Figure 8: *Frame time distributions*

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., AHERN S., KIRCHNER P. D., KLOSOWSKI J. T.: Chromium: A stream processing framework for interactive graphics on clusters, 2002. 1

[HM99] HEIRICH A., MOLL L.: Scalable distributed visualization using off-the-shelf components. In *IEEE Parallel Visualization and Graphics Symposium, pages 55–59, October 1999* (1999), IEEE. 2

[IG03] ISENBURG M., GUMHOLD S.: Out-of-core compression for gigantic polygon meshes. *ACM Trans. Graph. 22*, 3 (2003), 935–942. 6, 7

[LRN96] LEE T.-Y., RAGHAVENDRA C. S., NICHOLAS J. B.: Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers. In *IEEE Transactions on Visualization and Computer Graphics. 2(3)* (1996), IEEE, pp. 202–217. 2

[MEP92] MOLNAR S., EYLES J., POULTON J.: PixelFlow: High-speed rendering using image composition. *Computer Graphics 26*, 2 (1992), 231–240. 2

[MHS99] MOLL L., HEIRICH A., SHAND M.: Sepia: scalable 3D compositing using PCI Pamette. In *IEEE Symposium on FPGAs for Custom Computing Machines* (Los Alamitos, CA, 1999), Pocek K. L., Arnold J., (Eds.), IEEE, IEEE Computer Society Press, pp. 146–155. 2

[Mol91] MOLNAR S.: *Image-Composition Architectures for Real-time Image Generation*. PhD thesis, University of North Carolina, 1991. 1

[MPHK94] MA K.-L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications 14*, 4 (1994), 59–68. 2

[MWP01] MORELAND K., WYLIE B., PAVLAKOS C.: Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics* (2001), IEEE, pp. 85–154. 2

[NZ00] NGUYEN T., ZAHORJAN J.: Image layer decomposition for distributed rendering on nows. In *2000 International Parallel and Distributed Processing Symposium, Cancun, Mexico, May 1-5* (2000), IEEE. 2, 3

[Ols02] OLSON E.: *Cluster Juggler - PC cluster virtual reality*. Master's thesis, Iowa State University, 2002. 1

[Ora04] ORAD: Dvg technical presentation, http://www.orad.co.il/visual.htm, 2004. 2

[Rot02] ROTH M.: Integration paralleler renderingverfahren fuer lose gekoppelte systeme in opensg. In *OpenSG Symposium, Januar 2002* (2002), OpenSG. 1

[RS99] RAMAKRISHNAN C. R., SILVA C. T.: Optimal processor allocation for sort-last compositing under bsp-tree ordering. In *SPIE Electronic Imaging, Visual Data Exploration and Analysis IV* (January 1999), SPIE. 2

[SFLS00] SAMANTA R., FUNKHOUSER T., LI K., SINGH J. P.: Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware* (2000), ACM, pp. 97–108. 2, 7