Using Node and Batch Analysis to Efficiently Render Animations

Robert Foertsch, Brian M. Slator
Computer Science and Operations Research
North Dakota State University
Fargo, ND 58108
rjfoertsch@gmail.com, slator@cs.ndsu.edu

Abstract

The World Wide Web Instructional Committee (WWWIC) at North Dakota State University has been developing a high definition, stereoscopic animation depicting a Native American village in 1776. As this animation is in development, portions of the animation are rendered multiple times as changes occur.

Throughout repeated renderings of the animation, patterns began to emerge in relation to performance of specific machines used to render, and the time required to render complicated sections. With these observations, a new job dispatching method was developed to take these attributes into account, thus reducing the time required to render the animation.

1 Introduction

A distributed system can solve many real-world problems, but can also create new issues in terms of management and optimization, especially when the distributed system is built from heterogeneous hardware over a period of years.

This paper focuses on an optimization solution useful within the scope of animation projects our group works with, based on knowledge of previous rendering jobs and benchmarked information from the available nodes. This information allows us to allocate a specific number of frames to each available node, reducing inefficiencies that existed in our original system.

2 Background

The World Wide Web Instructional Committee (WWWIC) at North Dakota State University (NDSU) is an ad hoc committee of faculty, staff, and students working to advance education through the use of Immersive Virtual Environments (IVE; Slator et al. 2006).

One of WWWIC's current projects involves the creation of a twelve minute, stereoscopic animation of a Native American village. This village is called On-A-Slant, and existed near where the Heart River joins the Missouri River. We use Maya 3D modeling software to create and edit the models used in the animation, and in total, the complete animation will consist of over 36,000 frames.

For our purposes, the full animation is seldom rendered in its entirety. Changes are made to the animation, those portions of interest in the animation are rendered into individual frames, encoded into a video, sent out among the interested parties, improvements are suggested to the modelers, and more changes are made. This process continues, and we see portions of the animation rendered repeatedly with small changes occurring.

The scope of the project and the complexity of the models guarantee that a render job could not be completed in a reasonable amount of time on a single workstation. If a change is made to the animation, it is unrealistic to wait a week (or longer) to render the change and receive feedback. Thus a system was built to distribute the task of rendering the frames to many different machines. At the moment, WWWIC has a collection of older, decommissioned machines configured in a Beowulf cluster. In addition to this, various departmental machines are used, and when available we utilize CCAST: NDSU's Center for Computationally Assisted Science and Technology, who operate a number of high performance computing clusters.

To coordinate all of these resources, a central head node was created, along with a series of scripts to organize the machines to render the portions of the animation.

This original system consisted of a queue with a database backend maintaining the state of the machines. The head node has a list of frames that need to be rendered, and queried the database to see which machines were available. A frame is assigned to each machine, the machine renders the frame, and when complete, updates the central database stating that the frame is ready for retrieval and the node is ready to receive another frame.

In general, this system works fairly well. Machines continued to receive frames to render until the batch is complete. It is not a complicated set up, but there are numerous inefficiencies in the system. Also, complete logs of each rendered frame were kept, and patterns began to emerge that helped show a better method could be used to render the frames.

2.1 Queue Inefficiencies

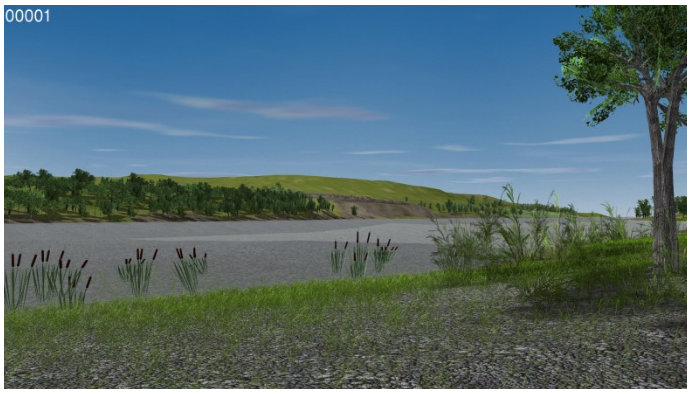
Allocating one frame at a time introduces much inefficiency. Maya allows the user to specify a range of frames to render. When implementing the queuing system, that range is one frame. Thus, for every frame rendered, Maya is needed to load the necessary binaries, models, textures, and other resources it needs for every frame rendered. Loading can take between 20 to 30 seconds. So, using the original queuing system, at least 20 seconds would be used for every single frame rendered. Assuming one of the nodes would render 400 frames in one batch; this single inefficiency can add over two hours to the overall render time. Instead, if Maya was specified the range of frames to render at runtime, the inefficiency resulting from the necessary resources being loaded would only occur once.

In addition to issues caused by Maya, other inefficiencies occur within the system. The head node needs to contact a machine for every frame rendered; this connection may take a second or two per frame. The worker node is configured to confirm that it received the message to render correctly. Post processing occurs, taking care of some error checking and log file parsing to be loaded into the head node’s database, a node may wait in a queue for seconds while the head node is contacting a different node to work, or sleeping before checking to see if any more nodes are ready to render; all these small communication connections between the head node and worker node add up to more inefficient uses of time, amplified as these occur for every frame rendered. Instead, if Maya is focusing on a range of frames to render, these could be taken care of elsewhere or in parallel while Maya is rendering the next frame. Such measures will be able to further reduce the time required to render a batch of frames.

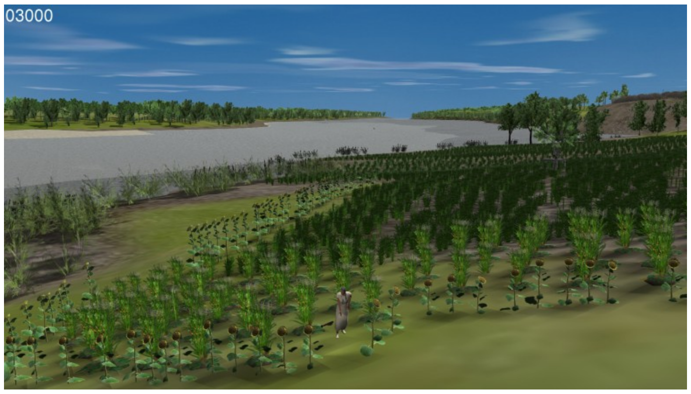
2.2 Frame Complexity

Through repeated rendering jobs of the animation, we have noticed a distinct pattern emerge in regards to the time that is required to render certain frames compared to others. A particular frame that consists of a wide expanse, with many different surfaces and textures in view takes a greater amount of time to render than another frame with few textures and surfaces in view of the camera, while keeping the hardware identical.

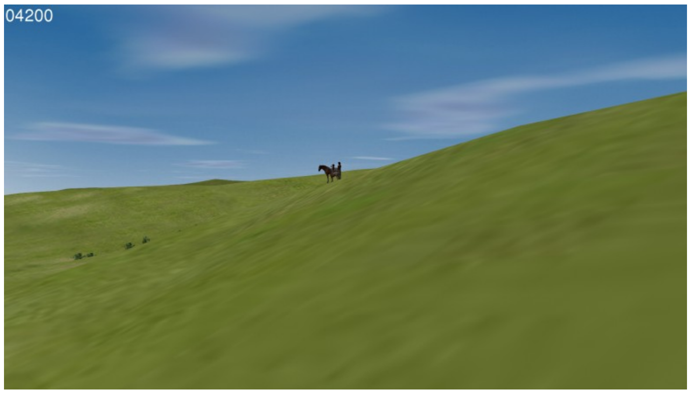
For example, Figure 1 shows three different types of frames, and the associated time it takes for the frames to render.



Frame 00001: This is a medium complexity frame. Many of the polygons are far in the distance. This frame takes 285 seconds to render on an Intel Core 2 Quad machine.



Frame 03000: This is a higher complexity frame. The large number of polygons associated with each plant in the garden adds up to create a complicated frame to render, in addition to the wide expanse in view. This frame takes 420 seconds to render on an Intel Core 2 Quad machine.



Frame 04200: This is considered a simple frame. The polygons simply consist of the ground and sky, and not many polygons are in view in the distance, as compared to frame 03000 and frame 00001 as mentioned before. This frame takes 53 seconds to render on an Intel Core 2 Quad machine.

Figure 1: three frames from different sections of the movie show the contrast between simple, medium, and higher image complexity.

In the context of an animation, we generally don't see large changes in frame complexity between consecutive frames. Since one second of video is 30 frames, any sort of dramatic shifts in the scene would not be visually appealing in our animation. Such dramatic shifts could occur with very rapid changes in camera path or angle; these cases do not occur in our current project.

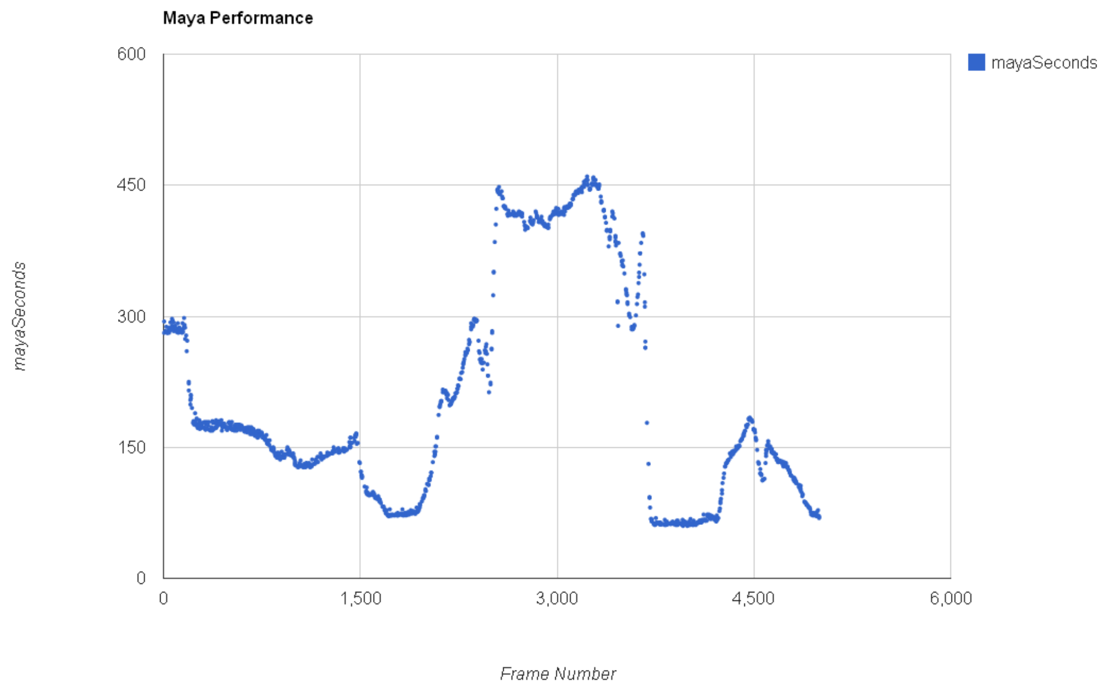


Figure 2: A scatterplot of rendering times for individual frames shows the variable difficulty of frames throughout the movie. The first 1700 frames are a traversal up a river, with relatively low render times. After 1700 frames the camera turns inland over more complicated terrain and animated figures that add significantly to the complexity of each frame.

To emphasize these observations, Figure 2 shows the relationship between the time required to render a frame and that frame's position within the greater scope of the animation, holding the capability of the machine on which these frames are rendered constant.

For calculating the frame complexity, we simply render the frames on hardware with similar processing ability. If similar hardware was not used, we would have two different variables in play: frame complexity and node ability. However, rendering all 5000 frames of the part of the animation that we are interested in on one machine would take approximately 13 days of render time. Fortunately, we have twelve machines available with identical hardware specifications (Intel Core 2 Quad processor with 2.66 GHz processor clockspeed and 4GB of memory). In addition, since the changes between consecutive frames are gradual, we do not benchmark every single frame. Instead, to achieve our frame complexity calculation, we render every fourth frame, and interpolate the frame complexity. Using multiple machines with similar hardware and interpolation of frame complexity allows us to reduce the time to benchmark the frames to about seven hours.

2.3 Node Ability

In addition to a pattern emerging between different frames of the animation, we can also show that a machine performs consistently when given a frame to render. This should come as no surprise. It has also led to the observation that two machines, with completely different hardware, consistently perform the same, in relation to each other, no matter which frame each node is working on.

To show these results, we will compare two machines, an Intel i7 2.67GHz with 6GB of memory with an Intel Pentium 4 1.90GHz with 512MB of memory against different frames.

Frame Number	Intel i7 Average Time (seconds)	Intel i7 Time Standard Deviation	Pentium 4 Time (seconds)	Pentium 4 Time Standard Deviation	Intel i7 Time / Pentium 4 Time
50	179.8475	0.8988	3325.625	17.8601	0.054079
1800	34.0065	1.9754	749.8718	13.0171	0.04535
2577	287.0317	0.4594	5914.571	28.1357	0.04853
3577	177.7619	0.5046	3567.256	14.2554	0.049832
4000	26.9397	1.1102	591.8056	14.2806	0.045521
4469	108.0282	0.3946	2188.4	15.2635	0.049364

Table 1: a comparison of render times on two platforms each with significantly different processing power.

Obviously the Pentium 4 machine requires more time to render a frame compared to an Intel i7 machine. The important piece, though, is that the ratio comparing the Intel i7 to the Intel Pentium 4 stays fairly consistent. Thus, regardless of which frame each machine is working on, assuming the frames have similar complexity, the nodes will perform comparably (and predictably) similar to each other.

3 Implementation

Foertsch et. al. 2011 described an idea on how to completely eliminate the queuing system. As noted, we wish to avoid allocating one frame to one node at a time. The head node would estimate an optimal dispatch, which can be built from knowledge gathered from previous rendering of those frames, in addition to benchmarked information about each node available. Each node is allocated a group of consecutive frames such that the sum of the computed work for that group of frames is compatible with the amount of work that node can accomplished in the calculated amount of time. Machines that have superior hardware will be allocated more render units (not necessarily frames) than machines with inferior hardware.

The following is an exercise allocating six frames between two nodes. For the sake of simplicity, we define one render unit as one minute of CPU time for one of the Intel Core 2 Quad machines we used to benchmark the frames. For example, the first frame of the animation requires 262 seconds to render, resulting in a frame complexity of 4.38 render units.

Once we know the total number of render units in the current batch, we can calculate how many render units of work each node will render, based on the node performance numbers calculated earlier. This tells us exactly what portion of the total amount of render units must be allocated to each node. Equation 1 shows the generic case.

$$\frac{1}{\mathrm{node}_1\mathrm{RU}} * t + \frac{1}{\mathrm{node}_2\mathrm{RU}} * t + \cdots = \mathrm{RU\ Total}$$

Equation 1: the general formula used to calculate the amount of rendering units each node will be allocated, along with the total time required for rendering those frames.

Host			Seconds per RU			
Intel Core 2 Quad			60			
Intel Pentium 4			180			
Frame Number	1	2	3	4	5	6
Render Units	2	3	3	4	2	1

Table 2: a simple example, using five frames and two render nodes.

The calculations work out as the following. We first calculate the sum of the combined render units of each frame, and solve for the time required to render.

$$\frac{1}{60} * t + \frac{1}{180} * t = 15$$

Equation 2: the specific formula used in the current demonstration.

We find *t* to be 675 seconds, meaning that with an optimal distribution of frames, rendering should complete in 675 seconds. Now we calculate the work of each individual unit (time * seconds per RU), and we find the Intel Core 2 Quad will render 11.25 RU and the Intel Pentium 4 will render 3.75 RU. However, this is an imperfect measurement; a frame is a discrete unit that we cannot split into portions of a frame to render. For the sake of simplicity, we find it best to allocate one frame over the allotted amount of render units. Rendering one less than the allotted amount of render units would leave frames unrendered near the end of the animation, which is unacceptable.

Frame Number	1	2	3	4	5	6
Render Units	2	3	3	4	2	1
Host	Intel Core 2 Quad				Intel Pentium 4	
Cumulative RU per Host	2	5	8	12	2	3

Table 3: allocating a consecutive set of frames to each node one frame at a time.

Thus the Intel Core 2 Quad will be dispatched to render frames one through four, and the Intel Pentium 4 will render frames 5 though 6.

4 Experiment

For the optimized dispatch to function, we run the exact same set of calculations as in the example above. For this paper, we are focusing on frames 1-4998, which comes out to being approximately 15015.35 render units. We will distribute these 4998 frames among the nodes using the benchmarked performance figures determined earlier.

In general, the metric we are interested in is the time it takes for the system to render the necessary frames, comparing our original method for dispatching frames (one frame at a time) vs. allocating a block of frames to render per node at the beginning of the dispatch job.

An experiment was conducted using the original dispatching method. The time required to render 4998 frames dispatching the frames one at a time was 19 hours, 35 minutes, and 24 seconds. This will serve as our control measure for the following experiment.

The following table shows the results of our intelligent dispatch, depicting the hostname, the lower and upper bounds for the frames to render, and the time required to render those frames.

Host	Starting Frame	Ending Frame	Total Render Time
lab19.cs.ndsu.nodak.edu	1	226	15h 19m 5s
lab18.cs.ndsu.nodak.edu	227	595	15h 4m 29s
lab17.cs.ndsu.nodak.edu	596	1026	15h 7m 9s
lab16.cs.ndsu.nodak.edu	1027	1500	15h 3m 11s
lab15.cs.ndsu.nodak.edu	1501	2113	11h 37m 1s
lab13.cs.ndsu.nodak.edu	2114	2377	15h 12m 21s
lab12.cs.ndsu.nodak.edu	2378	2574	15h 19m 33s
lab11.cs.ndsu.nodak.edu	2575	2715	15h 24m 51s
lab10.cs.ndsu.nodak.edu	2716	2861	15h 26m 57s
lab09.cs.ndsu.nodak.edu	2862	3005	15h 20m 11s
lab08.cs.ndsu.nodak.edu	3006	3143	15h 14m 57s
lab03.cs.ndsu.nodak.edu	3144	3335	Process died after frame 3190

lab01.cs.ndsu.nodak.edu	3336	3567	14h 59m 41s
lab00.cs.ndsu.nodak.edu	3568	4433	14h 57m 58s
Serenity	4434	4494	18h 19m 30s
192.168.1.8	4495	4520	14h 38m 26s
192.168.1.7	4521	4556	15h 32m 37s
192.168.1.5	4557	4594	15h 17m 55s
192.168.1.26	4595	4621	15h 15m 13s
192.168.1.25	4622	4653	15h 24m 7s
192.168.1.22	4654	4705	16h 5m 10s
192.168.1.21	4706	4739	16h 4m 3s
192.168.1.2	4740	4862	16h 4m 51s
192.168.1.19	4863	4944	15h 16m 1s
192.168.1.18	4945	4998	8h 55m 41s

Table 4: twenty-five machines of varying capacities are assigned sequences of frames to render according to their complexities with aggregate running times recorded.

As can be seen, every machine finished in less time than the queuing method, with the worst performing machine (Serenity) finishing one hour and 16 minutes sooner.

It should be noted that in this experiment, lab03.cs.ndsu.nodak.edu's Maya process died partway through the rendering process. This can happen no matter which rendering method was being used. A second process could be started on the worker nodes to detect whether a process died before completion, and restart the process with a command to being where it left off. This would add the time required to load Maya, and any time lost while working on the frame it crashed on (in this case, 3191).

5 Remarks

5.1 Experiment Summary

Experiment Name	Method	Time (Worst Case)
Control	Queue – Dispatch one frame at a time	19h 35m 24s
Experiment 2	Intelligent Dispatch	18h 19m 30s

Table 5: comparison of the two rendering methods.

In general, the results are promising. This experiment shows that by reducing inefficiencies through allocating frames in consecutive groups to each instance of Maya, we can decrease rendering times.

At this point in the project, one could look at the individual results for each node, and slightly alter the benchmarked value up or down, as needed. For example, in the experiment shown in Table 4, the Beowulf nodes were still rendering while all of the lab machines have completed, with some Beowulf nodes rendering a full hour after others have completed. It may be worthwhile to tweak the benchmarked values for the Beowulf nodes, adding seconds to the amount of time required for those machines to process one render unit.

192.168.1.18 was allocated fewer frames than its optimum amount of render units. However, this is to be expected. As mentioned earlier, each node is allocated slightly more than the optimal render unit amount, as we do not wish to have unrendered frames at the end of a batch. Thus one machine is expected to be underutilized, which is acceptable.

Lab15.cs.ndsu.nodak.edu finished in 11 hours and 37 minutes, which was quicker than the other lab machines (lab08 though lab19 all share identical hardware). This is inefficient, as it remained idle and unused while other nodes continued to render for many hours. A quick investigation revealed that when our node benchmark was ran, lab15 was experiencing hardware problems, and had its CPU speed downclocked for stability. When we conducted the experiment, the CPU speed had been increased to its original value. Once this machine is benchmarked again, we should see the overall render time of the intelligent dispatch reduced further.

Should more nodes become available, we would simply need to benchmark those machines to determine the amount of time required to process one render unit. Once that value is known, they can be added to the head node's optimal frame dispatch calculation.

5.2 Caveats

There are a couple caveats to the optimal dispatch method. First, information needs to be known about the frame complexity. At the beginning of an animation project, that information is not available. Another situation would be where we allow another group to use our rendering farm for their project. In these cases, a few batches would need to be rendered using the queuing method to generate the data required to make frame complexity assumptions.

Another issue is that this method is effective as long as the frame complexity remains similar between batches. In general, this is true. Many of the changes between different revisions of the animation may be as simple as moving the position of a tree from one point to another, or positions of modeled people, etc. Though these changes are useful in regards to the animation, they do not result in substantial changes in the frame complexity values of each frame. However, if we wished to speed up the camera path in a certain portion of the animation, this would result in different scenes being in view at different times, which could have drastic effects on the calculated frame complexity values. In this case, it may be worthwhile to render using the queuing method for at least one revision to rectify such changes. Analyzing the difference in times between nodes could be a simple indicator whether drastic changes to the frame complexity have occurred.

6 Conclusion

As we see, the main purpose is to show that within a rendering environment, information can be gathered about jobs and nodes to find the allocation of frames that utilize the available resources effectively. This general idea could be applied to other environments where information can be gathered about the complexity of jobs and the ability of each node.

7 Acknowledgements

The research described in this paper was supported in part by National Science Foundation Grant #IIP-0945807.

8 References

Foertsch, Robert, Matti Kariluoma, Brian M. Slator. (2011) 'Optimization of Job Efficiency in a Heterogeneous and Distributed Beowulf Cluster through Node and Job Analysis', paper presented at the *Midwest Instruction and Computing Symposium*, College of St. Scholastica, Apr 2011.

Slator, Brian M., Richard Beckwith, Lisa Brandt, Harold Chaput, Jeffrey T. Clark, Lisa M. Daniels, Curt Hill, Phil McClean, John Opgrande, Bernhardt Saini-Eidukat, Donald P. Schwert, Bradley Vender, Alan R. White. (2006). Electric Worlds in the Classroom: Teaching and Learning with Role-Based Computer Games. New York: Teachers College Press. Columbia University. 192 pages.