Out-of-Order Execution for Avoiding Head-of-Line Blocking in Remote 3D Graphics

John Stavrakakis and Masahiro Takastuka

[1] ViSLAB, Building J12 The School of IT, The University of Sydney, Australia
[2] National ICT Australia, Bay 15 Locomotive Workshop,
Australian Technology Park, Eveleigh NSW, Australia
jstavrakakis@vislab.usyd.edu.au, masa@vislab.usyd.edu.au

Abstract. Remote 3D graphics can become both process and network intensive. The Head-of-Line Blocking(HOLB) problem exists for an ordered stream protocol such as TCP. It withholds any available data from the application until the proper ordered segment arrives. The HOLB problem will cause the processor to have unnecessary idle time and non-uniform load patterns. In this paper we evaluate how the performance of an immediate mode remote 3D graphics system is affected by the HOLB and how the out-of-order execution can improve the performance.

Keyword: Distributed rendering, Network graphics, Load balancing.

1 Introduction

Interactive multimedia applications over a network demand both realtime delivery and excellent response time for a high quality end user experience. Networked applications using 3D graphics have difficulties ensuring this level of quality is maintained. The biggest problem is the sheer volume of network traffic they would generate. For audio and video multimedia, this is greatly reduced through appropriate down-sampling and discardment of perceptually unimportant data. In addition, they are able to exploit lossy network protocols that can continue without all the data[1]. This scenario is contrast to that of 3D graphics, as the the majority of data be preserved correctly which would otherwise cause severe artifacts to appear in the rendered images. Specifically, the complexity of graphics data creates greater network traffic and thus making it difficult to maintain desirable rendering frame rates. Networked 3D graphics has been well researched for high network utilisation, compression [2][3] and several techniques in reducing computational load [4]. Despite the attention, another bottleneck existing in the network graphics systems occurs between the passing of data from the network layer to the rendering engine. As the data is received from the network in the form of segments (a one-to-one correspondence to a packet sent/delivered), the segments are held in buffers until reassembly of the application data can take place. Following the reassembly of fragments to a segment, the segment will need to meet an ordering requirement prior to being passed on for rendering. Such a process is typically handled between decoupled networking and rendering modules within the system.

This problem is also known as Head-of-Line Blocking, and exists in the Transmission Control Protocol[5] (TCP) protocol. It occurs when a TCP segment is lost and a

D. Mery and L. Rueda (Eds.): PSIVT 2007, LNCS 4872, pp. 62–74, 2007.

subsequent TCP segment arrives out of order. The subsequent segment is held until the first TCP segment is retransmitted and arrives at the receiver[6].

This is not a problem for most software using TCP, as the requirement is not real-time and operation order is essential. Immediate mode graphics works differently; each command is simply defining the rendered scene, this requires no order until the drawing takes place. As such, 3D graphics is able to avoid the HOLB problem as its operations need not execute in order. The advantage of doing so allows the the graphics processor to avoid idling when data is available to execute.

This paper investigates the avoidance of the HOLB for a subset of applications using immediate mode 3D graphics. The following section will briefly introduce motivation for remote 3D graphics and currently available systems. Section 3 will detail the out of order nature in 3D graphics and its theoretical impact. We follow with experimental methods and results to address HOLB, and finally conclude with a brief summary of our findings and future work.

2 Remote 3D Visualisation Systems

Remote visualisation allows users to interact with graphics data over a network. There is a need for remote visualisation as it may be more convenient by location, more efficient by compute power to process data remotely, or to aid the infrastructure for remote collaboration such systems include AVS[7], Collaborative VisAD[8] and Iris Explorer[9]. Following our previous work, our concern lay within network graphics technology, which is transparent to the application.

In order to enable existing 3D applications to be made available in this fashion, there are two approaches: modify the application source to internalise the network component; or, extract the graphics from the application *transparently*, and then send it over the network. Example of systems belonging to the first approach can be found in a survey paper by Brodlie[10]. Advantages and disadvantages of these approaches were discussed in our previous work[11]. For reasons beyond the scope of this paper we look at the latter method.

Known methods for achieving this remote graphics distribution are typically done using OpenGL[12] an immediate mode graphics library. Immediate mode graphics asks the application to hold the graphics data and (re)transmit the data to the rendering engine to render the 3D graphics into a 2D image(frame). This type of remote rendering method is ideal for highly dynamic scenes where no simple predefining of the entities can be described or updated efficiently. Such examples include the rendering of large animated models[13] and also scale with fluid simulation[14].

GLX. A standard technique for remote display of 3D applications is GLX [15], the "OpenGL Extension to the X Window System". The X Window System[16] (often referred to as X) was developed to provide a network transparent user interface with rendering on a remote server. The X protocol is a client-server protocol in which applications are run on the client machine and display requests are sent to a server. This design allows application data and processing to be performed locally with window management and drawing to be handled transparently at the server. It is highly portable

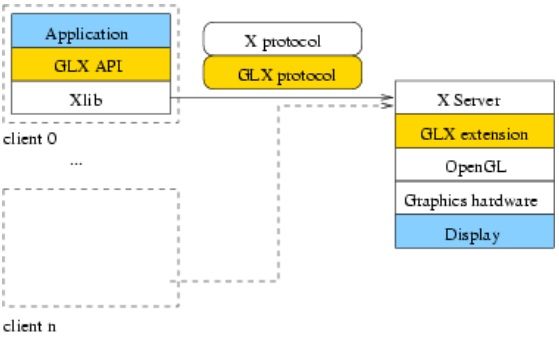


Fig. 1. GLX is composed of a client-side API, a protocol for GL command stream encoding, and an X server extension (all components shown in orange). The application resides on the client machine and the display is connected to the server (both indicated in light blue).

because any X client can connect to any X server, regardless of platform or operating system. Moreover, since all communication is handled by the client-side X library (Xlib) and the X server, applications do not have to be network aware.

GLX enables OpenGL applications to draw to windows provided by the X Window System. It is comprised of an API, an X protocol extension, and an X server extension. When using GLX for remote rendering, GL commands are encoded by the client-side API and sent to the X server within GLX packets. These commands are decoded by the X server and submitted to the OpenGL driver for rendering on graphics hardware at the server. Importantly, GLX provides a network transparent means of remote rendering to any X server that supports the extension. It also specifies a standard encoding for GL commands.

Figure 1 illustrates GLX. An application that uses the GLX API can send GL render requests to a remote X server that supports the GLX server extension. GL commands are encoded in the GLX packet, which is itself inserted into an X packet. Any number of clients can connect to an X server, but a client will only ever connect to a single X server. GLX is limited because of these characteristics. Rendering is always necessarily server-side and it cannot support GL command streaming to multiple remote displays. However, GLX is important because it establishes the fundamental capabilities required for OpenGL command streaming for remote rendering.

Chromium. Chromium is a well established and widely used distributed rendering system based on another technology called WireGL [17]. One of the major advantages of Chromium is that it enables users to construct a high-performance rendering system, which is capable of large scale complex rendering. Moreover, it can drive a large multi-display system to display high-resolution images. However, it was not designed to support sharing of 3D graphics following a dynamic producer/subscriber model.

Figure 2 illustrates Chromium's distribution model. It is general node-based model for stream processing. A node accepts one or more OpenGL command stream (GL stream) as input and outputs a GL stream to one or more other nodes. Each node contains one or more Stream Processing Units (SPUs) that modify the GL stream.

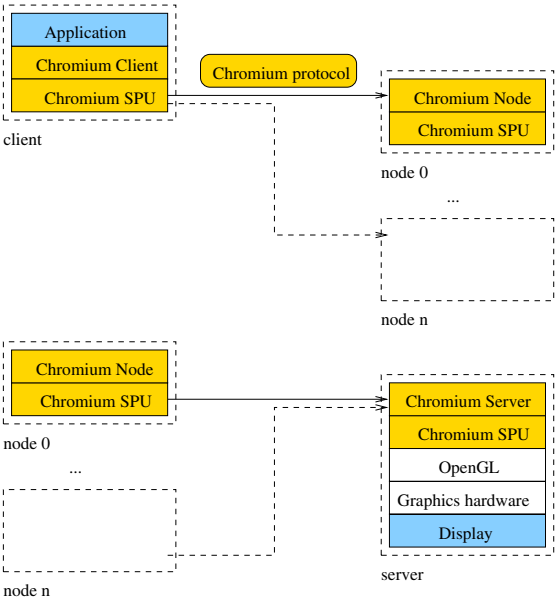


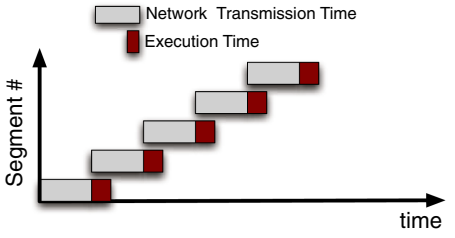
Fig. 2. Chromium has a flexible configuration system that arranges nodes in a directed acyclic graph (DAG). Each node can take input from, and send output to, multiple nodes. A client node takes GL commands from an application and creates a GL command stream. A server node takes GL commands from another node (and usually renders the stream on local hardware). Because Chromium is usually used for remote display, this diagram shows rendering and display at a server node. However, it is important to note that rendering (and display) can occur at the any node.

Rendering or display can occur at any node in the graph. This depends entirely on whether the SPUs on the node perform rendering or display. One characteristic of Chromium that is not illustrated in the figure is that configuration of the graph is centralized and set at system initialization. This is suitable for dedicated clusters (with fixed, single-purpose resources), but not ideal for grid computing (with heterogeneous resources that are dynamically added and removed, and also available for other services).

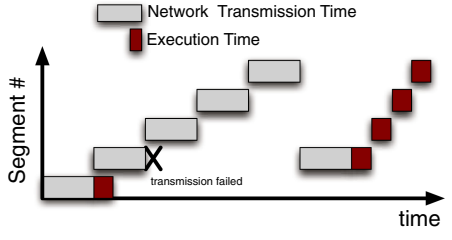
Chromium follows the OpenGL Stream Codec (GLS)[18] to encode GL commands. GLS defines methods to encode and to decode streams of 8-bit values that describe sequences of GL commands invoked by an application. Chromium, however, employs its optimized protocol to pack all opcodes into a single byte.

As the underlying delivery mechanisms of both Chromium and GLX rely on a TCP-like streaming protocols. The head of line problem becomes a greater issue when the amount of data increases, hence causing more network traffic. This would result in data being withheld for longer periods due to segment losses.

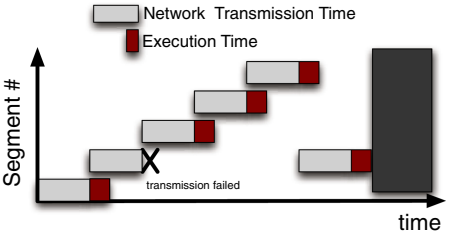
Such segment losses are illustrated by figures 3(a)-3(c). In figure 3(a), the system does not experience any packet loss. It must wait for the entire segment to arrive before it can execute it. Moreover, the processing of that segment can overlap with the retrieval



(a) No packet loss.



(b) Packet loss with ordering restraint.



(c) Packet loss with no ordering restraint.

Fig. 3. Figure 3(a) shows the system where no packet loss occurs, the system must wait for each packet to transfer and it can overlap processing intermediately. Figure 3(b) shows what will happen in an ordered sequence, the delay of execution in one segment will cause the processing of other segments to be delayed. Finally, figure 3(c) shows the unordered model, allowing any ready segments to be executed. The dark region denotes the time that processing was unnecessarily deferred to.

of the next segment. Figure 3(b) demonstrates what will happen when a segment fails to arrive in an ordered sequence. The execution delay of that single segment will forbid the processing of other segments until the first it has completed execution. Finally, figure 3(c) shows the effect of packet loss when no ordering is imposed on the execution of each segment. As a segment is received the only barrier to prevent its execution, is contention for graphics processing resources. Note the dark region denotes the time that processing was unnecessarily deferred to.

3 Out of Order Execution

An immediate mode graphics library can execute most commands out of order purely because, within rendering a frame, the commands being sent are defining the scene. The actual rendering of the final image requires all data to be present, thus requiring no specific ordering to define it in. This implies that synchronisation must at least occur at the frame boundaries, and was also found true for parallel graphics systems in order for the rendering to to maintain ordered consistency[19].

Data can be defined in this way due to the rasterisation process. In the pipeline processing view of the data, it is treated independently and is typically processed in parallel with one another until a later stage. It is only until the latter stage in the render-

ing engine, which would require intermediate results to be processed in order to satisfy dependencies between them. An example of such is depth sorting.

Unfortunately, not all commands can be executed out of order. The transformations, colours and textures that define an object need to be specified prior to executing the commands to define the object. This limits the applicability of out of order execution to either: components within an object sharing the same state, or whole objects that can be defined in with state very easily (discussed later).

We have chosen to use Lumino[20], a distributed rendering framework to transmit and execute OpenGL commands over a network. In a typical OpenGL application there are commands to define the beginning and end of a sequence of polygon data, these are glBegin and glEnd respectively. Within this Begin/End clause the expected data can take several formats, they can be points, lines, triangles or quadrilaterals[1]. No one command defines a polygon, rather we identify the sequence of commands as a *component*. The component is independent to any other component. As a polygon it may contain applicable state information such as a normal, color and texture coordinates. It is independent to other components since there are no shared vertices and any differing state information is available.

For example in defining a triangle mesh, the receiving end would simply obtain the segment containing a whole number of triangles, and immediately pass it on to the graphics processor. This is advantageous as the event of losing a segment, which contains other triangles, will not uphold the next arriving segment of triangles to be executed. It is also important to note that any transformations, depth ordering, lighting calculations etc. can be done during the reception of the remaining data of that frame rather than when the ordered segment has been delivered. The alternative to OOO contiguous blocks are independent blocks that can be executed independently with respect to one another. This case would appear when a burst of packets for an object are missed, but segments are available and hold enough state information to allow execution. This situation is quite complicated and is not considered in this study at this time.

The network communication was implemented as a reliable ACK-based UDP protocol. The advantage of doing so allows us to control the packet loss probabilities, segment sizes, window size (for flow control) and fine control over command alignment in the data packing of segments. Additional information that was packed in the header was a byte that indicates the Out-of-Order(OOO) sequence number. This number increments when switching between applicable OOO blocks and non-OOO blocks. Alternatively, the OOO segments could be determined by the receiver alone. However, it would be difficult to detect the appropriate OOO regions if segments are dropped and arrive out of order. By using a sender side method, the barriers between OOO and non-OOO sections are understood directly.

Let us consider a 3D model viewer program. For every frame that is rendered, the model is retransmitted from the source location be it system memory or file to the graphics processor. The primary geometric data will be included in Begin/End clauses, the size of these clauses scales with the geometric complexity of the data. For example,

[1] We isolate the study to independent components. GL_LINE_STRIP, GL_TRIANGLE_STRIP and GL_QUAD_STRIP cannot be executed out of order simply because there needs to be a reference point.

a reduced Stanford bunny model[21] has over 17,000 vertices. For the application to display the full model, all vertices are sent via begin/end clauses.

In one frame of rendering commands the size of the geometric data clearly dominates the remaining data (such as transformation matrices, lighting, miscellaneous items). The vertices alone account for: 17,000 * (12+2) = 232KB. Adding a normal for each triangle incurs an additional 77KB. The bandwidth required to retransmit this every frame at 24 frames per second is 7.3MB/s. As uncompressed data, there are circumstances by which there is not enough processing capacity to apply compression like that of Deering[22].

4 Experiment

The aim of the experiment is to observe the cumulative sum of waiting times from segment arrival to segment execution. We implemented this functionality as a plugin to Lumino using the aforementioned custom reliable UDP protocol. The sending side would identify the appropriate out-of-order sequences and selectively push component data into individual segments, as well as the necessary header information. The receiver would obtain each segment in no particular order and either withhold or execute segments. This depended on the out-of-order mode being disabled or applicable/enabled.

The sender is also configurable for packet loss probability, as well as the flow control window size and the segment size, which had remained fixed. The receiving end performs statistical monitoring of the arrival and the *beginning* of when a segment is to be executed. Note that the actual length of the execution time was not recorded as the resulting time difference to execute the segment data was comparable to making the expensive system call to gettimeofday twice.

The application under test was a simple ply model loader based on the clean[23] library. The ply format is a 3D format developed at Stanford University. The program begins by shaping the camera to fit the model in view, the animation then commences where the model rotates in a predetermined fashion without any randomisation. Using the display lists option in this program changes the rendering method from immediate to retained mode. In retained mode the model is cached at the graphics processor, and would avoid all bandwidth limitations. It is not always possible to use a display list and for our purposes this was disabled.

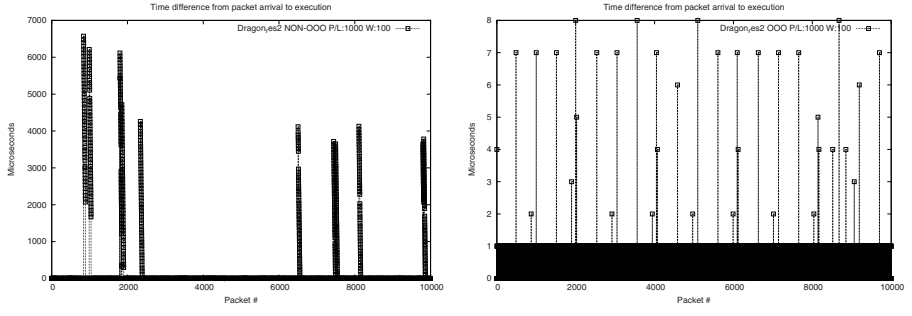
Tests were conducted between two machines within the local Gigabit LAN, the round trip time (RTT) between them yields the statistics min/avg/max/mdev = 0.076/0.108/0.147/0.014 ms. This test were all performed when no other users were active in the system.

Table 1. Machines used in tests

name	Specification
moth	Dual AMD Opteron 242 (1594MHz), 1GB DDR SDRAM Nvidia GeForce FX 5950 Ultra AGP. Ubuntu Linux 2.6.20 kernel.
cat	Intel Pentium 4 2.8GHz, 512MB DDR SDRAM , Nvidia GeForce FX 5700/AGP. Ubuntu Linux 2.6.20 kernel.

(a) Buddha. (b) Dragon. (c) Bunny17k.

Fig. 4. Screen shots of the model viewing application with the models chosen from by experiments.



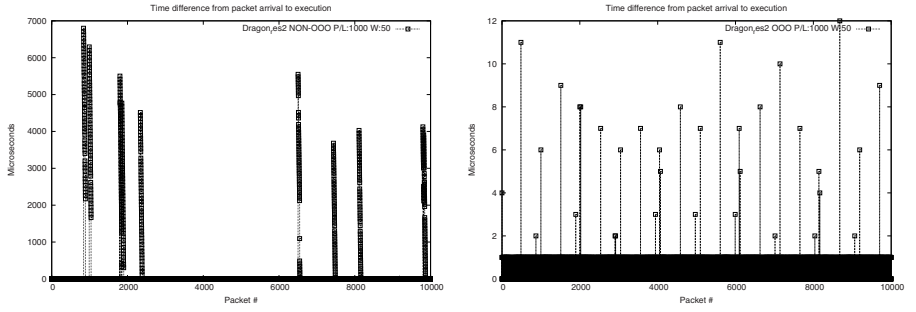
(a) Without OOO execution. (b) With OOO execution.

Fig. 5. The HOLB issue(left) is evident as the loss of single/burst of segments causes the execution delay of subsequent packets. The right shows an out of order protocol demonstrating that segment losses are isolated and independent to other segments.

The test models included decimated Stanford models: Dragon(res2), Buddha(res2) and a further reduced bunny (17k polygons) (see Figure 4. The use of the models in the experiment is to demonstrate how the large ooo segment can incur unnecessary delays in execution.

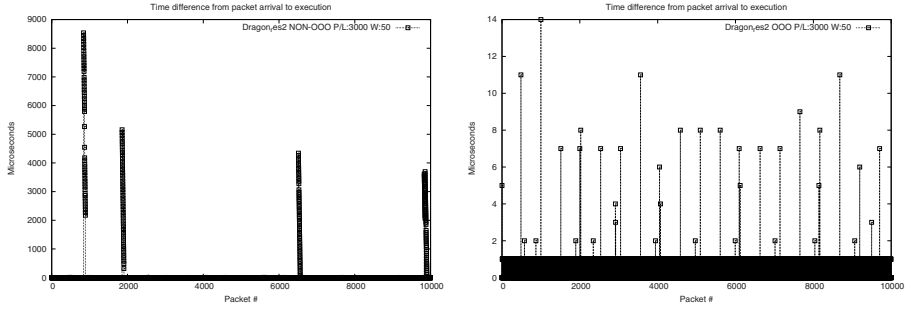
The pattern observed in figure 5(a) demonstrates the concept of head-of-line blocking problem. The single/burst of packets lost at one instant will not be delivered until there is a chance to access the network. The network in these cases are highly utilised, thus the waiting time is significant. At closer examination the vertical stripes have slope that is left to right, these are the segments immediately following the missing one, who are not able to continue, thus their waiting time is directly proportional to its 'packet' distance from the missing segment.

Figure 5(b) shows how an out of order execution model allows almost every packet to execute immediately on arrival. The actual delays visible here is the remainder of processing other out of order commands first. It is important to note that some segments require appropriate ordering, however, as the packet loss probability is 1 in a 1000 the chances to observe similar delay from 5(b) is too small.



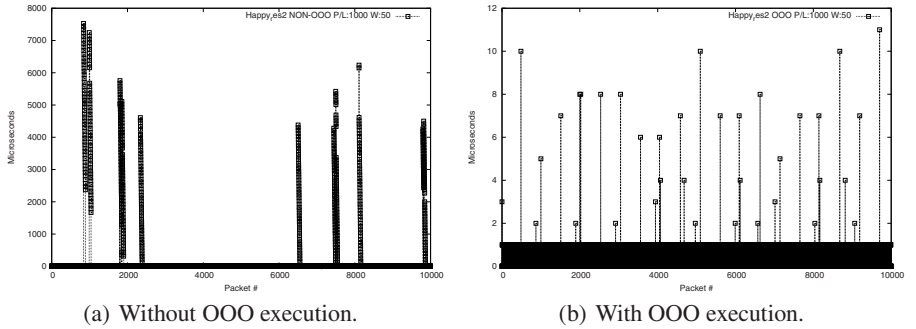
(a) Without OOO execution. (b) With OOO execution.

Fig. 6. The HOLB issue(left) is affected by sliding window like flow control methods. The larger the sliding window, the longer it will take before retransmission can take place. Either due to delaying the negative acknowledgement or by having to wait on network availability.



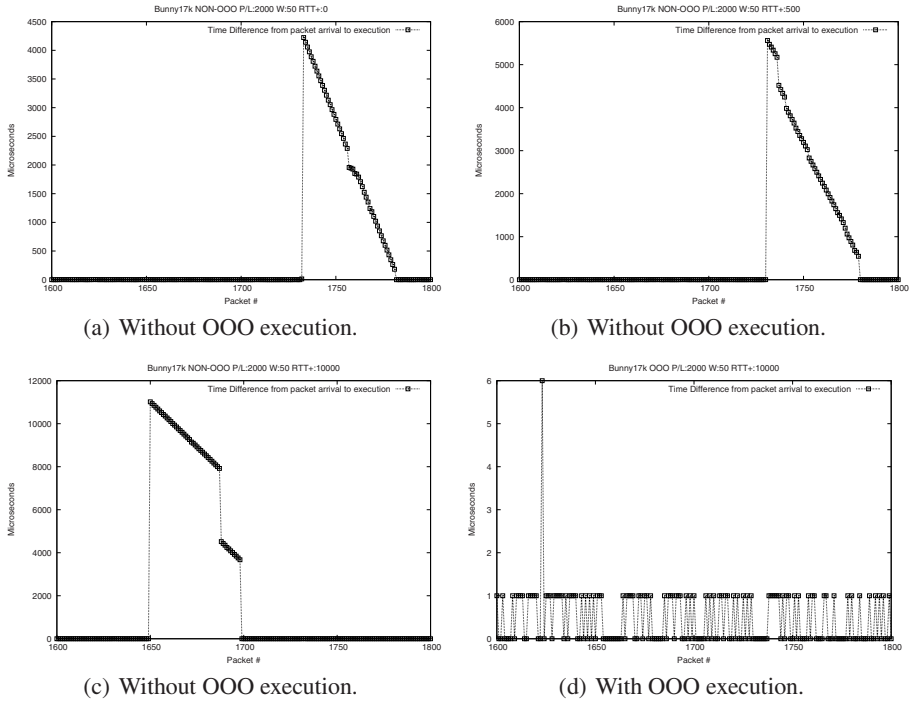
(a) Without OOO execution. (b) With OOO execution.

Fig. 7. The HOLB issue(left) is only an issue on packet loss. At this time the retransmission delay will cause a queue of unprocessed segments. The benefits of out of order execution appear much lower when segment loss is every $1/3000$.



(a) Without OOO execution. (b) With OOO execution.

Fig. 8. The HOLB issue(left) with a different model Happy Buddha. The large contiguous sequence of geometry helps avoid the HOLB problem.



(a) Without OOO execution. (b) Without OOO execution.

(c) Without OOO execution. (d) With OOO execution.

Fig. 9. This figure shows the time interval between delivery and execution for an increasing round trip time. The non OOO cases 9(a), 9(b), 9(c) have added delays of 0, 500, 10000 microseconds respectively. These areas denote the time that the segment was waiting for ordering to occur before being executed. The OOO case for delay 10000s is shown in 9(d), the effect of RTT does not interfere with the time between arrival and execution.

The influence of round trip time (RTT) is depicted in figure 9. A simulated delay for packet delivery was added for the non OOO cases: 9(a), 9(b), 9(c) as well as the OOO case:9(d). They each had an increased RTT of 0, 500, 10000 microseconds respectively. In the non OOO cases the distance between consecutive points (after a loss) represents the time that is taken to parse and process it, note that this gradient is machine dependent (machine cat4 used here). Such an area represents the time that the receiver was unnecessarily idle and/or the waiting on the ordering of segments for execution.

With zero additional RTT9(a), the delay is matching that of the protocol and the network alone, the response time in this case is within 4500 microseconds. Introducing a delay of 500 microseconds9(b) makes the effect of network delay visible. The recovery time worsened up to 6000 microseconds. The trend continues for a typical inter-city RTT(Sydney to Melbourne) time of 10,000 microseconds9(c). In this case, the stepping down represents the time at which the sender was recovering from the lost segment. During this time the sending window was full and no newer segments were being transmitted. It was only until the recovery occurred that the latter segments began to be transmitted, however as they arrived the ordering constraint on executing previous segments caused them to be delayed. The effects of HOLB will continue to ripple the

waiting time in this way for future arriving segments. Alternative protocols will offer various other distances to step down (denoting recovery), regardless, the loss of the single segment will incur an RTT penalty for consecutive segments within its own *sliding window* as shown in this protocol. Figure 9(d) is the OOO case where the RTT does not affect the interval between arrival and execution of the segment. It can be said to be invariant of RTT as it scales to larger networks (of higher RTT) for when out-of-order execution is applicable.

It is difficult to avoid the ordering constraint for non-contiguous segments as certain functions will change the state of OpenGL and would thus break the scene rendering. This is most apparent in parallel rendering systems such as Pomegranate[24], where there are several commands being executed simultaneously. This out of order execution occurs within the same one-to-one interaction between single consumer/producer model.

It is also important to note that the performance of rendering or network throughput is not a valid metric, as OOO execution in 3D graphics will *not* change the total time to transfer a frame of commands or to render them[2]. This is due to the frame synchronisation which requires all data to be available before the next segment of the newer frame can be executed. The observation made is that without OOO execution the load of the graphics processing unit appears non-continuous (surge of usage), however when it is used, load is continuous and more effective in competing for 3D graphics/CPU time share in the system.

5 Conclusion

By observing the flow of data in remote 3D graphics systems, we have discovered a potential bottleneck that can impact specific remote 3D applications. Our design decisions in building the experiment have been justified such that the nature of the bottleneck is clearly exposed. Using a controlled network protocol and large volumes of contiguous OOO graphics data, we are able to show that the HOLB problem presents unnecessary idle time for the graphics processor. This idle time is also dependent on the parameters of the network protocol, where increased sliding window sizes accumulated greater delay times for more segments, while single or burst of packet losses can still impact utilisation. We found that by having less restrictive ordering requirements, immediate mode graphics can alleviate the HOLB problem when the amount of geometric data becomes large.

To further exploit the HOLB with graphics, either: the specification of the graphics is required to change such that dependencies are satisfied in advance (moving towards retained mode) or the time of processing the segment data exceeds that of network costs. Such circumstances can exist when receiving systems are under a high load from competing 3D graphics clients, embedded devices may also take more time computing than utilising the network speed.

Our future work will aim at utilising this protocol for the two cases: load balancing in limited computing resources; and performing a dynamic trade off between idle process-

[2] The time taken to execute a segment was too insignificant to be considered an improvement, less the negligible overhead.

ing time to perform other functionality. For example, we could easily apply transparent compression on the components such that the bandwidth requirement is lowered, in doing so we raise the computational cost per segment and can move closer towards a more efficient network graphics system.

References

1. Liang, Y.J., Steinbach, E.G., Girod, B.: Real-time voice communication over the internet using packet path diversity. In: MULTIMEDIA 2001: Proceedings of the ninth ACM international conference on Multimedia, pp. 431–440. ACM Press, New York (2001)
2. Peng, J., Kuo, C.-C.J.: Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. In: SIGGRAPH 2005: ACM SIGGRAPH 2005 Papers, pp. 609–616. ACM Press, New York (2005)
3. Purnomo, B., Bilodeau, J., Cohen, J.D., Kumar, S.: Hardware-compatible vertex compression using quantization and simplification. In: HWWS 2005: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 53–61. ACM Press, New York (2005)
4. Varakliotis, S., Hailes, S., Ostermann, J.: Progressive coding for QoS-enabled streaming of dynamic 3-D meshes. In: 2004 IEEE International Conference on Communications, vol. 3(20-24), pp. 1323–1329 (2004)
5. Information Sciences Institute, U.o.S.C.: Transmission control protocol (September 1981)
6. Stevens, W.R., Fenner, B., Rudoff, A.M.: UNIX Network Progrmaming, vol. 1. Addison-Wesley, Reading (2004)
7. Advanced Visual Systems (AVS), Information visualization: visual interfaces for decision support systems (2002), `http://www.avs.com/`
8. Hibbard, B.: Visad: connecting people to computations and people to people (GET THIS). SIGGRAPH Comput. Graph. 32(3), 10–12 (1998)
9. Brodlie, K., Duce, D., Gallop, J., Sagar, M., Walton, J., Wood, J.: Visualization in grid computing environments. In: VIS 2004: Proceedings of the conference on Visualization 2004, pp. 155–162. IEEE Computer Society Press, Los Alamitos (2004)
10. Brodlie, K., Duce, D., Gallop, J., Walton, J., Wood, J.: Distributed and collaborative visualization. Computer Graphics Forum 23(2), 223–251 (2004)
11. Stavrakakis, J., Takatsuka, M.: Shared geometry-based collaborative exploratory visualization environment. In: Workshop on Combining Visualisation and Interaction to Facilitate Scientific Exploration and Discovery, British HCI, London, pp. 82–90 (2006)
12. Segal, M., Akeley, K., Frazier, C., Leech, J., Brown, P.: The opengl ® graphics system: A specification. Technical report, Silicon Graphics, Inc (October 2004)
13. Meta VR, Inc.: Meta VR virtual reality scene (2007), `http://www.metavr.com`
14. Liu, Y., Liu, X., Wu, E.: Real-time 3d fluid simulation on gpu with complex obstacles. In: Proceedings of 12th Pacific Conference on Computer Graphics and Applications, pp. 247–256 (2004)
15. Womack, P., Leech, J.: OpenGL® graphics with the X Window System®: Version 1.3. Technical report, Silicon Graphics, Inc. (October 1998)
16. The X.Org Foundation.: About the x window system, `http://www.x.org/X11.html`
17. Humphreys, G., Buck, I., Eldridge, M., Hanrahan, P.: Distributed rendering for scalable diaplays. In: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing, vol. 30, IEEE Computer Society Press, Los Alamitos (2000)
18. Dunwoody, C.: The openGL® stream codec: A specification. Technical report, Silicon Graphics, Inc. (October 1996)

19. Igehy, H., Stoll, G., Hanrahan, P.: The design of a parallel graphics interface. In: SIGGRAPH 1998: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, pp. 141–150. ACM Press, New York (1998)
20. Stavrakakis, J., Lau, Z.-J., Lowe, N., Takatsuka, M.: Exposing application graphics to a dynamic heterogeneous network. In: WSCG 2006: The Journal of WSCG, Science Press (2006)
21. Stanford University.: The stanford 3d scanning repository (2007)
22. Deering, M.: Geometry compression. In: SIGGRAPH 1995: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 13–20. ACM Press, New York (1995)
23. Real-time Rendering Group, The University of Western Australia. The clean rendering libraries (2005), `http://60hz.csse.uwa.edu.au/libraries.html`
24. Eldridge, M., Igehy, H., Hanrahan, P.: Pomegranate: a fully scalable graphics architecture. In: SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 443–454. ACM Press, New York (2000)