Speed up of rendering pipeline by deferred lighting and triple queue structure

B.-S. Liang and C.-W. Jen

Redundant operations and stalls prevent a rendering pipeline from full-speed operation. To speed up the rendering pipeline, a triple queue structure is proposed to smooth the pipeline and to obtain benefit from deferred lighting. The results of cycle-accurate simulation show that the proposed structure can reduce rendering cycles to 52.9% in small size queues.

Introduction: The rendering pipeline is a mainstream method to implement three-dimensional (3D) graphics rendering. However, the rendering pipeline method has two problems. The first problem is redundant operations on invisible polygons. Deferred lighting [1] can delete the redundant operations, and save operation cycles on them. However, if not well handled, the saved cycles may become useless 'bubbles' in the pipeline. Our purpose was to design a specific architecture to obtain benefit from the save cycles. The second problem is pipeline stalls. Due to various polygon sizes, polygon and pixel rates are not in linear proportion. Stalls occur whenever the data rates mismatch the process speed of hardware. A straightforward way to avoid stalls is to insert one queue as a buffer between the polygon and pixel rate areas [2, 3]. However, the queue size quickly grows if a longer queue is required, since 20 to 40 bytes are required for a single entry in this queue. To solve these two problems, we propose a triple queue structure to smooth the pipeline and to obtain benefit from deferred lighting. Since three queues surround the lighting module, bubbles and stalls in the pipeline can be avoided. Therefore, this design can obtain benefit from deferred lighting, and speed up the rendering pipeline significantly by only having small queue sizes.

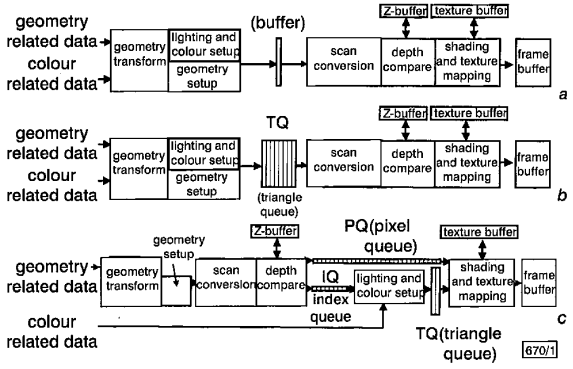

Fig. 1 *Architectures of rendering pipelines*

a Without queue
b Single queue structure
c Triple queue structure

Table 1: Queue lengths in different overall queue size

Overall queue size (bytes)		256	512	1K	2K	4K
Single queue structure	Triangle queue (TQ) 32 bytes/entry	8	16	32	64	128
Triple queue structure	Pixel queue (PQ) 2 bytes/entry	64	128	256	512	1024
	Index queue (IQ) 2 bytes/entry	32	64	128	256	512
	Triangle queue (TQ) 32 bytes/entry	2	4	8	16	32

Table 2: Cycle numbers in rendering pipeline without queue

Pattern name	Dolphins	Al	Al over Dolphins	Dolphins over Al
Long polygon cycle (50:1)	138434	602261	739160	736328
Short polygon cycle (10:1)	70845	209683	279534	272176

Triple queue structure: Fig. 1*a* shows a conventional pipeline without a queue, with only one buffer between stages. Fig. 1*b* shows the single queue structure. One queue, TQ (triangle queue), is inserted before scan conversion, at the border between the polygon and pixel rate areas. The single queue structure can be a reference to show the speed up by queue without deferred lighting. Fig. 1*c* shows our proposed triple queue structure. The lighting module, 'Lighting and colour setup', is deferred and sandwiched between two queues: IQ (index queue) and TQ. The IQ stores global indices of polygons before lighting, and TQ stores lighted polygon information. The lighting module acts as a delay. Conversely, another queue, PQ (pixel queue) stores pixels the parent polygons of which are in the IQ, the TQ or the lighting module. The length of the PQ is longer since each polygon may map to many pixels. However, the size of the PQ is still small since each entry is only two bytes for coded *xy*-co-ordinates of pixels.

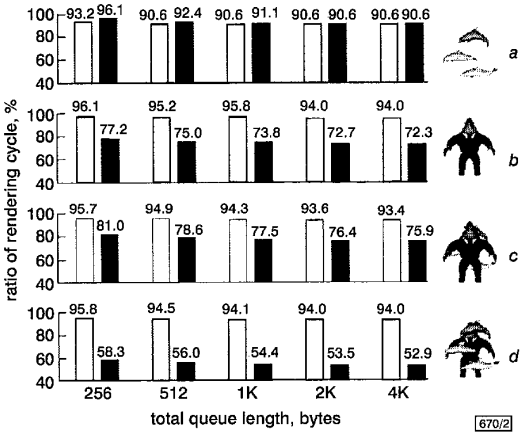

Fig. 2 *Ratio of rendering cycles in long polygon cycle (50:1)*

a Dolphins
b Al
c Al over Dolphins
d Dolphins over Al
□ single queue structure
■ triple queue structure
100% = rendering cycles in rendering pipeline without queue

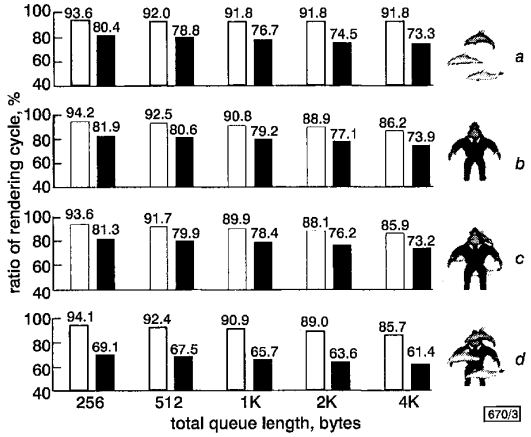

Fig. 3 *Ratio of rendering cycles in short polygon cycle (10:1)*

a Dolphins
b Al
c Al over Dolphins
d Dolphins over Al
□ single queue structure
■ triple queue structure
100% = rendering cycles in rendering pipeline without queue

Rendering in triple queue structure: As shown in Fig. 1*c*, to render a polygon in a 3D scene, geometry-related data of this polygon enter the rendering pipeline first. The polygon is then transformed and scan-converted into a group of pixels for depth comparison. If all pixels of this polygon fail in depth comparison, the polygon is invisible and will be discarded. No data will be sent to queues in this condition. In contrast, if any pixel passes the depth comparison, the *xy*-co-ordinates of all passed pixels are sent to the PQ. Meanwhile, the global index of this polygon is sent to the IQ. When the 'Lighting and colour setup' module receives the global

index from the IQ, it also receives the colour-related data by the global index. After lighting and colour setup, the lighted polygon information is queued in the TQ. The 'Shading and texture mapping' module then receives the lighted polygon information from the TQ and pixel co-ordinates from the PQ to generate shaded and textured pixels to construct the final image in the frame buffer. Since the pixel order of our triple queue structure is identical to the conventional one, rendering pipelines with the triple queue structure can realise transparency and alpha blending operations correctly.

Cycle-accurate simulation: We construct SystemC models of the rendering pipelines for cycle-accurately simulation [4]. Two kinds of conditions are considered; first, the long polygon cycle, its ratio of polygon cycles to pixel cycles being 50:1. We assume each polygon requires 150 cycles in the 'Lighting and colour setup' module, and each pixel requires three cycles in the 'Shading and texture mapping' module. The second condition is the short polygon cycle, the ratio being 10:1. Each polygon takes only 30 cycles in this case. In the triple queue structure, the 'Lighting and colour setup' requires an additional ten cycles to receive the colour-related data. The display resolution is 640 × 480. Four simulation patterns are: 'Dolphins', 'Al', 'Al over Dolphins' and 'Dolphins over Al'. The last two patterns are combinations of the first two, and the Dolphins are always prior to the Al in the rendering sequence. This enables back-face culling. The simulated queue sizes range from 256 to 4K bytes. In the single queue structure, all sizes are assigned to the TQ. In our triple queue structure, half size is assigned to the PQ, quarter size to the IQ and quarter size to the TQ. Each entry size is 2 bytes in the PQ for encoded pixel co-ordinates, 2 bytes in the IQ for the global index of polygons, and 32 bytes in the TQ for information of the lighted polygon. The queue lengths in different overall queue size are listed in Table 1.

Simulation results: We adopt the cycle number in the rendering pipeline without queue as criterion, as listed in Table 2. Figs. 2 and 3 show the results of long and short polygon cycle conditions, respectively. In the two Figures, the single queue structure is denoted by white bars. For the long polygon cycle condition in Fig. 2, the TQ queue is always empty and loses its function. Therefore, the rendering cycles remain long, no matter how large the queue size. For the short polygon cycle condition in Fig. 3, the single queue functions better, but the speed up is still limited. However, owing to deferred lighting, the triple queue structure shows better speed up, as shown in the black bars in Figs. 2 and 3. Especially in the 'Dolphins over Al' pattern in Fig. 2*d* and Fig. 3*d*, because the Dolphins are rendered earlier, many polygons in the following Al are invisible and discarded in depth comparison, and therefore the lighting module saves many operation cycles. Regrading the operations of the triple queue, the saved cycles can contribute to speeding up overall rendering. Hence, the rendering cycle can be widely shortened to 52.9%.

Conclusion: The pipeline stalls caused by data rate changes and the redundant operations on invisible polygons prevent full-speed operation of the rendering pipeline. Although the queue reduce pipeline stalls, the performance of the single queue structure is limited and only able to reduce rendering cycles to 85.7%. To avoid stalls and to benefit from the deferred lighting technique, we propose a triple queue structure to speed up the rendering pipeline. The cycle-accurate simulation results show that, with the same overall queue size, our structure achieves better performance than the single queue structure, and can reduce rendering cycles to 52.9%.

Acknowledgment: This work was supported by National Science Council, Taiwan, Republic of China, under Grant NSC-89-2218-E009-085.

11 July 2001
Electronics Letters Online No: 20010901
DOI: 10.1049/el:20010901

B.-S. Liang and C.-W. Jen (*Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China*)

E-mail: bsliang@ee.nctu.edu.tw

References

1 LIANG, B.-S., YEH, W.-C., LEE, Y.-C., and JEN, C.-W.: 'Deferred lighting: a computation-efficient approach for real-time 3-D graphics'. Proc. IEEE Int. Symp. on Circuits and Systems, 2000, pp. IV-657–IV-660

2 MONTRYM, J.S., BAUM, D.R., DIGNAM, D.L., and MIGDAL, C.J.: 'Infinitereality: a real-time graphics system'. Proc. Comp. Graphics (SIGGRAPH), 1997, pp. 293–303

3 HARRELL, C.B., and FOULADI, F.: 'Graphics rendering architecture for a high performance desktop workstation'. Proc. Comp. Graphics (SIGGRAPH), 1993, pp. 93–100

4 Open SystemC Initiative, 'SystemC source code and manual version 1.0.1', http://www.systemc.org/, 2001

Adaptive approach for FEC reliable multicast

T. Lestayo, M. Fernández and C. López

An adaptive integrated forward error correction approach to provide reliability in a multicast environment is proposed. The proposed solution avoids the undesirable feedback implosion phenomenon regardless of the number of receivers and uses minimum bandwidth in the feedback channel.

Introduction: One approach to improve the performance of reliable multicast transport protocols is to use jointly the capabilities of forward error correction (FEC) and automatic repeat request (ARQ). There exist two ways of combining FEC and ARQ. In the layered model [1], FEC operates independently beneath the ARQ layer. Here, the role of the FEC function is to reduce the packet loss probability seen by the ARQ layer, thereby decreasing the number of retransmissions and the network bandwidth requirements. In the integrated model [2], the FEC and ARQ functions are integrated into the same layer and the FEC module operates as follows: for a given set of data packets (also referred to as a transmission group (TG)), it computes and holds a set of redundant or parity packets. Once the sender has received all the requests across the feedback channel, it transmits the minimum number of parity packets needed to recover all the losses in the original data set. The efficiency of this protocol has been analysed in [2], where it is shown that integrated FEC can substantially reduce the amount of network bandwidth usage. The reported results also show that, for a large community of receivers, sending proactively a (small) number of parity packets along with the data ones is not detrimental to efficiency and could reduce feedback. However, this number depends on factors such as the number of receivers and the loss probability along the path from the source to each receiver, which is not always completely known to the sender. Therefore, in real network operations, the number a of parity packets initially sent should vary as the multicast group size or the network load change [3].

In this Letter, we propose an algorithm to estimate the optimal number of initial parity packets with prior knowledge of neither the population size of the multicast group nor the transmission conditions inside the network. We describe a proactive integrated FEC/ARQ protocol that uses this algorithm to dynamically vary the proportion of initial parity packets, and we study the performance of this technique.

Bandwidth analysis: The proposed algorithm will be incorporated into a multicast protocol that obeys these generic rules:

(i) The sender multicasts k data packets along with a parity packets. The value of a is selected to provide the minimum overall bandwidth. Later, we will provide the details of the proposed adaptive selection mechanism.

(ii) Each receiver requests in unicast mode the number of parity packets that are required to recover the entire TG, if any are required.

(iii) The sender multicasts the highest number of parity packets requested. Steps (ii) and (iii) are repeated until all users recover the TG.

The total bandwidth consumed by the protocol can be decomposed into that of forward (from sender to receivers) and back-