RenderWare Graphics

# White Paper

---

# PS2 Timing Diagrams

Copyright © 2003 – Criterion Software Ltd.

# Contact Us

## Criterion Software Ltd.

For general information about RenderWare Graphics e-mail info@csl.com.

## Developer Relations

For information regarding Support please email devrels@csl.com.

## Sales

For sales information contact: rw-sales@csl.com

## Acknowledgements

With thanks to the RenderWare Graphics development and documentation teams.

The information in this document is subject to change without notice and does not represent a commitment on the part of Criterion Software Ltd. The software described in this document is furnished under a license agreement or a non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or non-disclosure agreement. No part of this manual may be reproduced or transmitted in any form or by any means for any purpose without the express written permission of Criterion Software Ltd.

Copyright © 1993 - 2003 Criterion Software Ltd. All rights reserved.

Canon and RenderWare are registered trademarks of Canon Inc. Nintendo is a registered trademark and NINTENDO GAMECUBE a trademark of Nintendo Co., Ltd. Microsoft is a registered trademark and Xbox is a trademark of Microsoft Corporation. PlayStation is a registered trademark of Sony Computer Entertainment Inc. All other trademark mentioned herein are the property of their respective companies.

# 1. Timing Diagrams

## Conventions

In the diagrams that follow the vertical lines represent VSYNC interrupts and so divide the horizontal axis (time) into frames. Each diagram contains two rows of boxes. The upper row represents activity on the CPU, and bottom row activity on the VU1/GS. The CPU activity is broken into application updates (represented by the boxes labeled with a U) and rendering, labeled with R. The rendering here refers to RW building DMA packets in memory, rather than the actual drawing of primitives. The drawing is shown in the lower row as the boxes labeled with G. (We have bundled together VU1 microcode processing and GS drawing). The number in each box is just a sequence number.

## Starting the Application

In all the diagrams we deliberately show that the first application update happens at an arbitrary time during the first frame. Since the CPU is not synchronized to the VSYNC interrupt and the application can start at an arbitrary time, the position of the first box is also arbitrary.

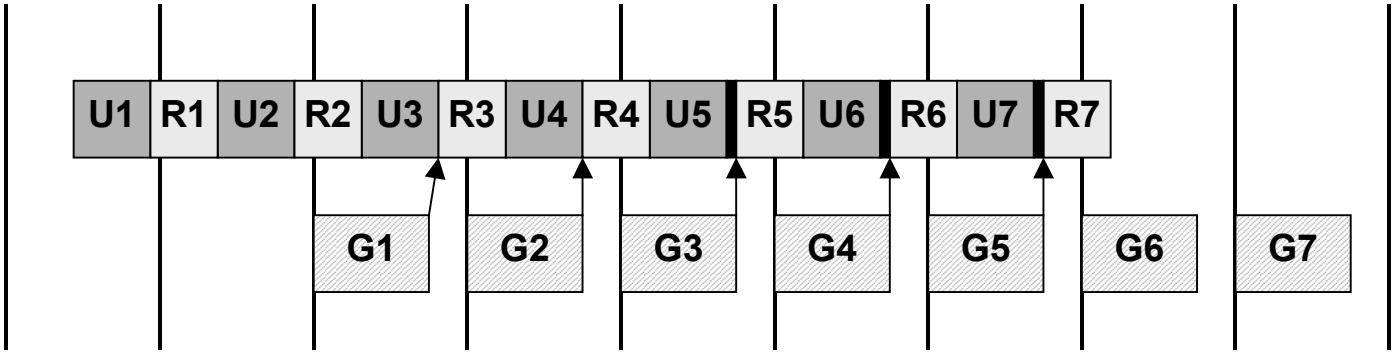
## Synchronization and Parallelism

The key to understanding these diagrams is to see the activity on the CPU and VU1/GS as totally separate. The CPU spends all the frame time performing AI/physics/game logic, which we lump together as application update, followed by some time rendering. These two activities can take as much time as the programmer decides is necessary. Obviously to render at 60Hz, this CPU time must take less than 1/60$^{th}$ of a second. When RW is rendering, all that it is doing is building DMA packets in memory. From RW3.4, there is a single, double buffered memory pool. This means that RenderWare can only ever have a maximum of two frames queued up (one in each buffer. A single frame might span both buffers if insufficient space was allocated.). A typical scenario is that one DMA list, representing frame N is being DMAed to the VU1/GS whilst RenderWare is building frame N+1 in memory. By the time that RenderWare comes to build frame N+2, frame N is transferred, and so the buffer used by frame N is available for storing frame N+2[1]. The RenderWare API call `RwCameraClear` marks the division between application update and rendering stages. If this function detects that there is no free buffer in the DMA memory pool, then it will wait for the DMAC to finish transferring a previous frame. (This busy waiting represents "wasted" time on the EE, and appears in the diagrams below as the dark, shaded area. Applications can register a callback to consume this time if required.) A buffer will become free following a DMAC interrupt that marks the end of the previous transfer. In the diagrams that follow, this interrupt is shown as an array connecting a G box to an R box. The rendering stage cannot occur until this interrupt happens indicating that a buffer is available. Rendering of frame N+2 cannot start until the display of frame N has finished. Note that the start of DMA activity transferring a frame from a DMA buffer in EE memory to the VU1/GS always starts on a VSYNC.

[1] Note the subtle distinction between a buffer becoming available and space being available in the buffer. RenderWare never considers the amount of memory in the pool that is free, the decision is based only on whether one of the two buffers is empty.

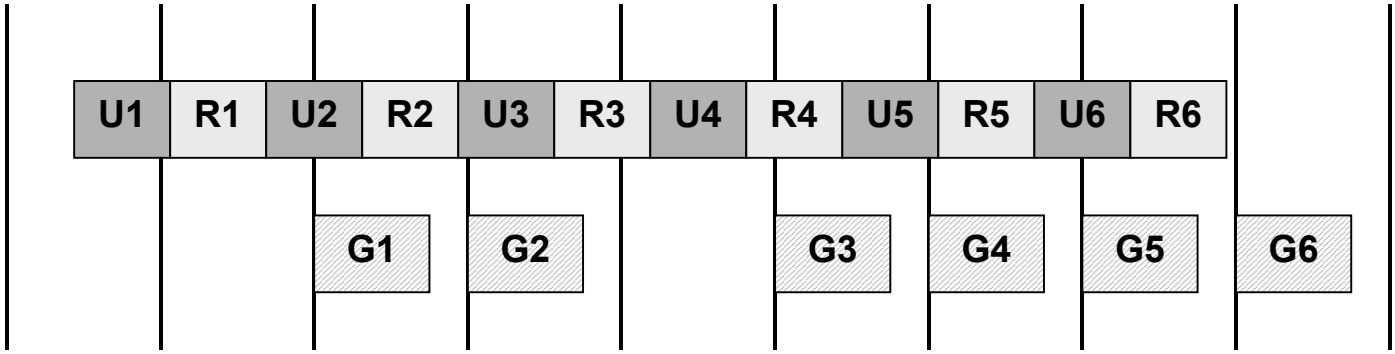
# 2. Scenario One – Well-balanced

The diagram below illustrates the case when the application takes just under 1/60th of a second to complete the generation of a frame, and the VU1/GS also complete inside the frame time. You can see a very small amount of wasted time from frame 5 onwards. Note that the CPU activity is not synchronized to the VSYNC interrupt, but, instead, is synchronized to the DMA interrupt indicating that a previous frame has been transferred. This is totally unexpected to most RW users. Obviously in the diagram the time spent actually drawing the triangles could be made bigger. Also in this case it would be possible to synchronize the EE to the VSYNC interrupt.



# 3. Scenario Two – CPU Bottleneck

In this case the time spent on the EE updating and rendering is longer than 1/60th. This indicates that the CPU is the bottleneck. Since the dispatch of data to the VU1/GS is faster than the generation of a new frame, the EE never has to wait for a buffer to become available. The game will drop frame-rate, and this can be seen as the gap in processing between frames 2 and 3. The application could dispatch less geometry to the GS and still not make 60Hz.



# 4. Scenario Three – VU1/GS Bottleneck

In this case the generation of DMA packets is considered to be fast, but the time it takes to draw the frame on the VU1/GS is too long to make 60Hz. The frame-rate then drops to 30Hz as the figure shows. It appears from the figure that the CPU and VU1/GS activities no longer overlap greatly. This might be taken to mean that there is little parallelism in the system being exploited. The explanation for this is that the time spent on the EE is much less than 1/30$^{th}$ of a second. The rendering and update stages could be greatly extended and so overlap more with the DMA transfers and VU1/GS activity.