RenderWare Graphics

White Paper

PS2 Metrics

Contact Us

Criterion Software Ltd.

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

Contributors

RenderWare Graphics development and documentation teams

Table of Contents

1. Introduction

Metrics Libraries

During development of a title that uses RenderWare Graphics you will want to investigate the performance that you are achieving. The RenderWare Graphics SDK provides special libraries to help you with this task. These libraries are called the "metrics" libraries and are peers of the release and debug libraries. The metrics libraries are almost identical to the release libraries that you will use for the final gold of your title. They differ in that they record statistics within the default RenderWare Graphics rendering pipelines. Although there is an overhead associated with collecting this information, it is very small. You could quite easily use the metrics libraries throughout the development of your title and not the release libraries (bear in mind that you should also use the debug libraries extensively during your project).

Visualization of Metrics

The metrics libraries record statistics during execution of your application. Values are available for things such as the number of triangles shipped to the hardware per frame. Note that the metrics libraries do not display this information. Instead, the metrics data is stored in an `RwMetrics` structure. You can use the API call `RwEngineGetMetrics()` to obtain a pointer to the structure that RenderWare Graphics populates with data. The examples that are shipped with the SDK can be built against metrics libraries. A preprocessor symbol, `RWMETRICS`, must be defined for any applications into which these libraries are linked. In the skeleton code that the examples use, this symbol also causes the code in `metrics.c` to be compiled into the project. This source file contains a routine called `RsMetricsRender()` that displays the metrics data over the 3D display. This function also makes use of the PS2-specific metrics rendering routine called `psMetricsRender()`. If your application is not using the skeleton then you can copy and paste these functions into your project since RenderWare Graphics contains no built-in functions to display metrics. Alternatively, you may wish to display metrics using bar graphs (using immediate mode) or record the information to a disk file for offline processing.

2. Entries

The metrics information gathered on PlayStation 2 is described below.

numTriangles

This relates to the number of triangles that are being drawn per frame. If you loaded an atomic which contained 10,000 triangles (as returned by `RpGeometryGetNumTriangles()`) and rendered it then you would see this entry returned as 10000. (Assuming that the object was not totally outside the frustum, in which case the atomic would be culled and not sent to the GS for rendering.) This metric therefore returns useful numbers for artists. The `numTriangles` metric includes immediate mode triangles. It might be possible that this metric records two additional triangles. These are used for darkening the screen area where the skeleton displays metrics.

numProcTriangles

To improve rendering performance of tri-strip models, RenderWare Graphics adds additional triangles to their geometry. These triangles have zero area and the GS uses this fact to decide that they do not need to be drawn. These zero-area triangles are known as "degenerate" triangles in RenderWare Graphics terminology. The `numProcTriangles` metric records the number of triangles sent to the GS, including these degenerates but excluding extra passes. If you use tri-strip models – something that is highly recommended – you will notice that this metric is higher than the `numTriangles` metric.

numExtraPassTriangles

This metric records the number of triangles sent to the GS due to the extra passes used by some pipelines. The sum of this and numProcTriangles gives the total number of triangles sent to the GS.

numVertices

This metric simply records the number of vertices that are DMA'd from PlayStation 2 MAIN memory to VU1 each frame. Note that the degenerate triangles added by RenderWare Graphics do *not* insert additional vertices. `numVertices` does not increase when clipping occurs. Vertices (and triangles) introduced by clipping exist only on VU1 and are not counted by the metrics libraries.

numResourceAllocs

When a retained-mode object (e.g. an atomic) is first drawn, RenderWare Graphics will create DMA packets that are used to upload its vertex data to VU1. These packets are re-used for subsequent renders of the object and so do not need to be rebuilt (unless the object is modified or the resources arena runs out of space). This improves performance, since creating DMA packets is a slow operation. (As a very rough guide it might take one hundred times as long to create a DMA packet as it takes to re-use it.) The memory used for storing these DMA packets of vertices is taken from a pool called the *Resource Arena*. The resource arena, therefore, stores vertices in the preferred format for PlayStation 2. This conversion from the platform independent representation used for objects to the platform specific format is called *instancing* in RenderWare Graphics terminology. When the application starts up, it can determine the size of this memory pool (using `RwResourcesSetArenaSize()`). This pool is used like a cache, so if RenderWare Graphics needs to instance a new object and no space is left in the resource arena it will evict the DMA packet used for uploading another object. The `numResourceAllocs` metric records the number of times per frame objects require instancing. As with any caching technique it is possible for thrashing to occur in the resource arena. This will show up as a large `numResourceAllocs`. (See the White Paper on *Dynamic Memory Management on PlayStation 2* in RenderWare Graphics).

numTextureUploads

The GS video memory can store between approximately 1.5 and 2.5 Megabytes of texture (varies according to the video mode selected). RenderWare Graphics uses this video memory area as a cache. When a texture is needed for rendering that is not currently in the cache, RenderWare Graphics will find space in the video memory and set up a DMA transfer to copy the texture from main memory to the GS. The `numTextureUploads` metric records the number of times per frame that this occurs. It represents a count for all textures that are uploaded. There is no way for the metrics library to record how many times each texture is uploaded.

sizeTextureUploads

This metric records how many bytes of texture data are uploaded from system memory to GS video memory per frame. This does not include the memory for the DMA and GIF tags that are used during the transfer.

VU1 utilization

RenderWare Graphics uses VU1 for all geometry processing, including lighting, clipping, back-face culling, object-to-camera space transformations and camera projection. The `VU1 utilization` metric records the percentage of time per frame for which VU1 is busy processing geometry.

DMA1 utilization

RenderWare Graphics uses the DMA1 channel for uploading geometric data to VU1. The `dma1 utilization` metric records the percentage of time per frame for which the DMA1 channel is busy transferring geometry. The same VU1 sampling technique is used for this metric as for `VU1` utilization.

DMA2 utilization

RenderWare Graphics uses the DMA2 channel for uploading textures to the GS. The `dma2 utilization` metric records the percentage of time per frame for which the DMA2 channel is busy transferring texels to the GS video memory. Additionally, RenderWare Graphics uses DMA2 for transferring render state changes. This will contribute a very small amount to this metric, probably less than 1%. The usual VU1 sampling technique is also used for this metric.

VSyncs between flips

The `Vsync between flips` metric records the frame rate. On the PlayStation 2 it represents the number of VSYNC interrupts that have occurred since the front and back buffers were swapped. The metric therefore can be used to determine the frame rate. If you divide 60 by this metric you will get the frame rate, so a metric of 1 indicates 60Hz rendering, and a metric of 2 indicates 30 (=60/2) Hertz rendering. Due to the quantization inherent in this metric, determination of the "real" frame rate (that which would be measured in the absence of VSYNC) is hard to determine, though averaging the quantized frame rate over several frames may produce a reasonable figure.

CPU utilization

The application can determine how much free main CPU time is available by using this metric. It is computed as the time spent not waiting for a frame DMA buffer to become free. (RenderWare will only wait on this if the application has managed to generate an entire frame while the previous frame is still dispatching, and then a Camera usage function is called for the following frame.)

3. Interpreting Metrics Information

Geometry

If you use tri-strip models (which is recommended on PlayStation 2) then the ratio between `numProcTriangles` and `numTriangles` can give an indication as to how well the artwork has been modeled. If RenderWare Graphics has had to add many degenerate triangles to connect separate short tri-strips together then these additional triangles (which contribute nothing to the visual appearance, but which can improve performance) will act as an overhead. RenderWare Graphics adds approximately 4 degenerate triangles to connect tri-strips. The pathological case is where it is impossible to form any tri-strips within the geometry of a particular model. In this case, RenderWare Graphics will add four degenerate triangles for every triangle in the model. The ratio between `numProcTriangles` and `numTriangles` would then reach a value of 4. This indicates poor artwork, not suitable for tri-stripping (e.g. a particle system). The artwork should be reworked to improve tri-stripping or should not be tri-stripped at all. Note that RenderWare Graphics contains many tri-stripping algorithms and you should try all of them, and use the algorithm that seems to work well with your artwork. Remember that the presence of triangle fans prevents artwork from tri-stripping, and also that it is not possible to tri-strip across material boundaries. Further, if you use shading groups in 3ds max, bear in mind that this will force RenderWare Graphics to duplicate vertices along the boundaries between groups (since each vertex will have more than one normal). This also prevents a tri-stripper from connecting triangles across the boundary (adjacent triangles in a tri-strip must reference the same vertices for the edge that they share).

The `numVertices` metric is not particularly useful. If it is clearly three times the `numTriangles` metric then is can indicate that the artwork is not tri-stripped. In this case the `numProcTriangles` metric will equal the `numTriangles` metric, since RenderWare Graphics will have added no degenerates.

Textures

RenderWare Graphics receives an interrupt for every texture that is uploaded to the GS video memory. For this reason, a large number of texture-uploads can significantly reduce the time the MIPS core has to perform processing. RenderWare Graphics uploads a texture, all of its mipmap levels and its palette (if it has one) in a single.

It is easy for the number of texture transfers to exceed the number of textures in the scene. This situation indicates that the texture cache is being thrashed – so the same texture is needed by different parts of the scene and has to be uploaded more than once. It is very important that the texture budget is carefully planned and 4- or 8-bit textures used wherever possible.

If you suspect that texture uploads are playing a large part in reducing your triangles-per-second count, then an easy experiment to perform is to remove all textures from the scene. This is most easily achieved by renaming the folder where the textures live. This will cause RenderWare Graphics to fail to read any textures from disk. Rendering with no textures should indicate what performance you can achieve if all of your textures fit in video memory at once. Unfortunately, there is a trade-off to be made between raw triangle throughput and how rich the textures are.

Note that the `RpMatFX` plugin could be used to compress textures. Any texture can be decomposed into a lower resolution RGBA texture and a 4-bit or 8-bit monochrome intensity map. Multi-pass rendering can be used to apply the color map first, followed the intensity map, using a multiplicative blend. This approach was described at the 2001 US Sony DevCon.

Processing

The VU1 and DMA utilization figures are sampled and not measured. This means that they should be used as a guide and not be taken literally. The time spent processing a vertex on VU1 is normally much longer than the time that it takes for the DMA hardware to transfer the vertex. This should mean that the utilization of DMA1 is lower than the utilization of VU1. Clearly the objective is to maximize both the DMA1 and VU1 figures. If you are using skinning, then a fair amount of work is performed on VU1 (4 matrix blends per vertex). This can mean that the VU1 utilization is large compared to the DMA1 utilization.

DMA2 utilization will be large if many textures are uploaded.