RenderWare Graphics

White Paper

Dynamic Memory Management on PlayStation 2

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. Platform-Generic Memory Allocation Mechanisms for RenderWare Graphics

The following section applies to RenderWare Graphics running on all supported platforms.

Free Lists

A RenderWare Graphics free list provides optimized dynamic allocation of items of size

- known at compile-time, and
- constant at run-time

The significant point here is that their count

- need not be known at compiler-time, and
- need not be constant over at run-time.

As the list is filled, the free list mechanism will automatically allocate more memory in developer-defined chunks.

Optimizing the performance of this mechanism can require some fine-tuning work, but the result should be an optimal trade-off between memory usage and speed.

Generic Garbage Collection with RwFreeListPurgeAllFreeLists()

The function `RwFreeListPurgeAllFreeLists(void)` performs garbage collection on all free lists in the system, returning the amount of memory freed in bytes.

This would typically be called in a natural pause during execution, for example when moving from level to level in a game, where the latency of the call is amortized with other house keeping.

Specific Garbage Collection with RwFreeListPurge(RwFreeList * freelist)

The function `RwFreeListPurge(RwFreeList * freelist)` performs garbage collection on the entire specified free list, returning the amount of memory freed in bytes.

This would typically be called in to free memory between calls to `RwFreeListPurgeAllFreeLists(void)` to release memory from lists which the application knows likely to have accumulated "dead wood" but without incurring the latency of full garbage collection over all free lists.

The Resource Arena

The Resource Arena is a RenderWare Graphics cache. Its size is passed as a parameter to `RwEngineInit()`.

During rendering, RenderWare Graphics stores platform-specific instances of model geometry within this cache. If the model is not changed substantially from one frame to the next, the instanced data is simply fetched and transferred to the rendering hardware, rather than being recomputed.

This system improves the speed of the rendering process, but the size of the Resource Arena needs to be fine-tuned for optimal results. A badly tuned Resource Arena will affect performance adversely.

Determining the Working Set Size with RwResourcesGetArenaUsage()

The function `RwResourcesGetArenaUsage()` returns the total amount of instance arena memory used in the most recently rendered frame. The number returned may be larger than the arena size if the arena is "thrashing"; this will cause performance to degrade and the application may wish to grow the arena to prevent this. Thrashing in this context means that the resource arena has run out of space. During rendering RenderWare Graphics needs space in which to store platform-specific rendering data, and to create this space another object is evicted from the cache. In the next rendering frame, this second object is no longer instanced, the cache is full, and so a third object is evicted. This behavior means RenderWare Graphics spends more time instancing geometry than rendering.

Note that the value returned by this function is only meaningful before a call to `RwCameraShowRaster()`, since the latter resets the usage count. Use this function directly before such a call, and after all rendering calls have been made.

Setting the Resident Set Size with RwResourcesSetArenaSize()

The function `RwResourcesSetArenaSize()` sets the size of the cache reserved for the resources instancing arena; the default size is that passed to `RwEngineInit()`.

If called between `RwEngineInit()` and `RwEngineStart()` to set the initial size for the arena before the memory is allocated.

If used at a later stage, the arena will be emptied and the memory reallocated.

2. PlayStation 2 - Specific Memory Allocation Mechanisms for RenderWare Graphics

The following section applies to RenderWare Graphics running on PlayStation 2.

Setting the Size of the PlayStation 2 DMA workspace with _rwDMAPreAlloc()

The size of the PlayStation 2 DMA double buffered workspace may be set with

```
rwcore.h: extern RwBool _rwDMAPreAlloc(RwUInt32 size,
RwUInt16 PURSize, RwUInt128 *optBuf);
```

where

- `size`
  is the workspace in bytes. It will be allocated 128 byte aligned, so the allocation call (if required) is actually size+0x80. Defaults to 1Mb.

- `PURSize`
  is the maximum number of procrastinated unrefs (Each is a pointer. The space is allocated from "size".) Defaults to 1K entries.

- `optBuffer`
  if non-null, a pointer to a 128 byte aligned buffer of "size" that the DMA system should use as its workspace, rather than allocating its own. Defaults to NULL.

This function should be called before calling `RwEngineStart()`

It will pre-allocate `memory` for dynamic data, including

- render state changes,
- DMA "glue", and
- immediate mode instance data

The amount required is hard to judge a priori, being extremely application dependent. RWDEBUG builds will print warning messages if a frame's worth of data cannot be double buffered and a buffer swap is forced.

If `RwEngineStop()` is called, the default sizes will be reset, so `_rwDMAPreAlloc()` must be called again if required.