Video Memory (VRam)

Video Ram (VRAM) and JavaFX

JavaFX applications tend to need significant VRAM to operate properly, particularly applications that use large images, but also for fonts and gradients.

On some machines VRAM is (or used to be) very fast special purpose memory. Currently on many systems, and on most embedded systems, VRAM is allocated (or reserved) from the main system memory. VRAM is consumed by the actual frame buffer (often with two copies allocated for quick swapping of the front and back buffer), and by GPU resources such as image/texture maps, as well as other system needs.

JavaFX contains a texture caching mechanism that attempts to work within a limit of VRAM. Unfortunately, there is not a standard means of determining how much VRAM is available to start, nor is there a way to estimate the efficiency of that allocation. The efficiency is similar to standard `malloc()` which employs bucket mechanisms but means that a small allocation will consume the nearest minimum bucket size.

The JavaFX texture caching mechanism currently defaults to 512 MBytes.

On desktop configurations, VRAM configuration is not normally an issue, and many systems share system and vram. For some intensive applications, changing the JavaFX cache size may boost performance.

For embedded devices the JavaFX default, which was chosen based on typical desktop configurations, may actually exceed the available memory. For example, the minimum recommended memory split for JFX on the Pi is 128 MBytes, with many applications requiring 256 MBytes.

Since the JavaFX texture caching mechanism currently defaults to 512 MBytes - a value that will likely exceed what is available on the Pi, it is quite possible that an image intensive application might fail when the cache exceeds the available system limit.

Remember that the allocated VRAM will also be consumed by each framebuffer (width * height * 2 byte per pixel * 2+ for swapping) as well as any other system needs.

The property setting `-Dprism.maxvram=90M` can be used to set the JavaFX texture cache limit, and in this example, setting it to 90 MB. This value would be a good starting value for a memory split of 256MB for VRAM.

To debug a JavaFX application failure to allocate VRAM, `-Dprism.poolstats=true` can be used to monitor the actual usage of the texture pool to better determine the upper limit.

Another property setting `-Dprism.targetvram=xx` is used to limit the amount of VRAM that the texture cache will use at any given time, freeing textures that are no longer likely to be re-used when this value is exceeded. Least used textures will be discarded, and recreated on need. If too many recently used textures would be discarded for a given target, the target is raised, subject to the maximum limit controled by the `maxram` property. The default for this setting is calculated as 1/8th of the `maxvram` setting, equal to about 11M for the example of 90M. This fractional setting works well for desktop applications, but may be a bit restrictive during the startup phases for some embedded applications. The system will naturally raise the target appropriately, but some experimentation with a larger value and `-Dprism.poolstats=true` may result in better startup performance. The property `-Dprism.verbose=true` can also be used to monitor when the target for the default texture pool is raised and a more appropriate initial target can be chosen for a given application.

Platform Specific Notes

i.MX6

The video memory is usually set using a kernel boot parameter. To check the kernel boot arguments, use:

```
# cat /proc/cmdline
```

looking for something like vmalloc=xxxM. Changing this value (or setting it if it its missing) is done by changing the u-boot script.

A different way to check for the current value is:

```
# root@nitrogen6x:~# dmesg | grep vmalloc
```

Raspberry Pi

Video Memory allocation is covered in the Raspberry Pi page.