Performance Guidelines

v. 0.4

1. Scenes

Per scene targets:

- 50k to 100k triangles
- 50k to 100k vertices
- 50 to 100 draw calls

An application may be able to render more triangles by using very simple vertex and fragment programs, minimizing overdraw, and reducing the number of draw calls down to a dozen. However, lots of small details and silhouette edges may result in visible aliasing despite MSAA.

It is good to be conservative! The quality of a virtual reality experience is not just determined by the quality of the rendered images. Low latency and high framerates are just as important in delivering a high quality, fully immersive experience, if not more so.

Keep an eye on the vertex count because vertex processing is not free on a mobile GPU with a tiling architecture. The number of vertices in a scene is expected to be in the same ballpark as the number of triangles. In a typical scene, the number of vertices should not exceed twice the number of triangles. To reduce the number of unique vertices, remove vertex attributes that are not necessary for rendering.

Textures are ideally stored with 4 bits per texel in ETC2 format for improved rendering performance and an 8x storage space reduction over 32-bit RGBA textures. Loading up to 512 MB of textures is feasible, but the limited storage space available on mobile devices needs to be considered. For a uniquely textured environment in an application with limited mobility, it is reasonable to load 128 MB of textures.

Baking specular and reflections directly into the textures works well for applications with limited mobility. The aliasing from dynamic shader based specular on bumped mapped surfaces is often a net negative in VR, but simple, smooth shapes can still benefit from dynamic specular in some cases.

Dynamic lighting with dynamic shadows is usually not a good idea. Many of the good techniques require using the depth buffer (expensive on a mobile GPU) with a tiling architecture. Rendering a shadow buffer for a single parallel light in a scene is feasible, but baked lighting and shadowing usually results in better quality.

To be able to render many triangles, it is important to reduce overdraw as much as possible. In scenes with overdraw, it is important that the opaque geometry is rendered front-to-back to significantly reduce the number of shading operations. Scenes that will only be displayed from a single viewpoint can be statically sorted to guarantee front-to-back rendering on a per triangle basis. Scenes that can be viewed from multiple vantage points may need to be broken up into reasonably sized blocks of geometry that will be sorted front-to-back dynamically at run-time.

2. Resolution

Due to distortion from the optics, the perceived size of a pixel on the screen varies across the screen. Conveniently, the highest resolution is in the center of the screen where it does the most good, but even with a 2560x1440 screen, pixels are still large compared to a conventional monitor or mobile device at typical viewing distances.

With the current screen and optics, central pixels cover about 0.06 degrees of visual arc, so you would want a 6000 pixel long band to wrap 360 degrees around a static viewpoint. Away from the center, the gap between samples would be greater than one, so mipmaps should be created and used to avoid aliasing.

For general purpose, rendering this requires 90 degree FOV eye buffers of at least 1500x1500 resolution, plus the creation of mipmaps. While the system is barely capable of doing this with trivial scenes at maximum clock rates, thermal constraints make this unsustainable.

Most game style 3D VR content should target 1024x1024 eye buffers. At this resolution, pixels will be slightly stretched in the center, and only barely compressed at the edges, so mipmap generation is unnecessary. If you have lots of performance headroom, you can experiment with increasing this a bit to take better advantage of the display resolution, but it is costly in power and performance.

Dedicated “viewer” apps (e-book reader, picture viewers, remote monitor view, et cetera) that really do want to focus on peak quality should consider using the TimeWarp overlay plane to avoid the resolution compromise and double-resampling of distorting a separately rendered eye view. Using an sRGB framebuffer and source texture is important to avoid “edge crawling” effects in high contrast areas when sampling very close to optimal resolution.

3. Hardware Details

The Adreno has a sizeable (512k - 1 meg) on-chip memory that framebuffer operations are broken up into. Unlike the PowerVR or Mali tile based GPUs, the Adreno has a variable bin size based on the bytes per pixel needed for the buffers -- a 4x MSAA, 32 bit color 4x MRT, 32 bit depth render target will require 40 times as many tiles as a 16 bit depth-only rendering.

Vertex shaders are run at least twice for each vertex, once to determine in which bins the drawing will happen, and again for each bin that a triangle covers. For binning, the regular vertex shader is stripped down to only the code relevant to calculating the vertex positions. To avoid polluting the vertex cache with unused attributes, rendering with separate attribute arrays may provide some benefit. The binning is done on a per-triangle basis, not a per-draw call basis, so there is no benefit to breaking up large surfaces. Because scenes are rendered twice for stereoscopic view, and because the binning process doubles it again (at least), vertex processing is more costly than you might expect.

Avoiding any bin fills from main memory and unnecessary buffer writes is important for performance. The VrLib framework handles this optimally, but if you are doing it yourself, make sure you invalidate color buffers before using them, and discard depth buffers before flushing the eye buffer rendering. Clears still cost some performance, so invalidates should be preferred when possible.

There is no dedicated occlusion hardware like PowerVR chips have, but early Z rejection is performed, so sorting draw calls to roughly front-to-back order is beneficial.

Texture compression offers significant performance benefits. Favor ETC2 compressed texture formats, but there is still sufficient performance to render scenes with 32 bit uncompressed textures on every surface if you really want to show off smooth gradients

`GlGenerateMipmaps()` is fast and efficient; you should build mipmaps even for dynamic textures (and of course for static textures). Unfortunately, on Android, many dynamic surfaces (video, camera, UI, etc) come in as SurfaceTextures / samplerExternalOES, which don't have mip levels at all. Copying to another texture and generating mipmaps there is inconvenient and costs a notable overhead, but is still worth considering.

sRGB correction is free on texture sampling, but has some cost when drawing to an sRGB framebuffer. If you have a lot of high contrast imagery, being gamma correct can reduce aliasing in the rendering. Of course, smoothing sharp contrast transitions in the source artwork can also help it.

2x MSAA runs at full speed on chip, but it still increases the number of tiles, so there is some performance cost. 4x MSAA runs at half speed, and is generally not fast enough unless the scene is very undemanding.

Last Update: Nov 10, 2014