PowerVR Performance Recommendations

The Golden Rules

Filename	:	PowerVR Performance Recommendations.The Golden Rules
Version	:	PowerVR SDK REL_16.1@3976567a External Issue
Issue Date	:	17 Mar 2016
Author	:	Imagination Technologies Limited

Contents

List of Figures

1. Introduction

1.1. Document Overview

The following document covers key principles developers should follow in order to avoid critical performance flaws in their graphics applications. These recommendations come from the combined experience of the PowerVR Developer Technology Support team and the developers they work with, profiling and optimising their applications and games during development.

1.2. Common Bottlenecks

It is important to understand where performance is bottlenecked before attempting to optimise an application. This ensures effort isn't wasted or visual quality isn't sacrificed for minimal gains. If an optimisation is inappropriately applied to an area that isn't bottlenecking performance there may be no performance gains and, in some cases, an incorrectly applied optimisation may lead to worse performance.

From the PowerVR Developer Technology team's experience, we have derived the following list of common bottlenecks generally found in unoptimized OpenGL ES applications, as ordered from most to least common:

- CPU Usage.
- Bandwidth Usage.
- CPU/Graphics Core Synchronization.
- Fragment Shader Instructions.
- Geometry Upload.
- Texture Upload.
- Vertex Shader Instructions.
- Geometry Complexity.

Profiling tools are vital in this process for developers to understand what is happening in their application, the hardware it is running on, and how and where bottlenecks are occurring. The PowerVR SDK includes the profiling tools PVRTrace and PVRTune to aid development on platforms powered by PowerVR hardware.

2. The Golden Rules

2.1. The Principle of ‘Good Enough’

This principle refers to the notion that “if the viewer cannot tell the difference between differently rendered images always use the cheaper implementation".

This is most commonly an issue for applications ported from desktop or console to mobile devices, where developers will use the same assets for the lower specification devices. When working with a smaller screen it is possible to greatly simplify shaders, meshes, or textures while maintaining perceived fidelity and therefore reducing graphics core workload without compromising visuals.

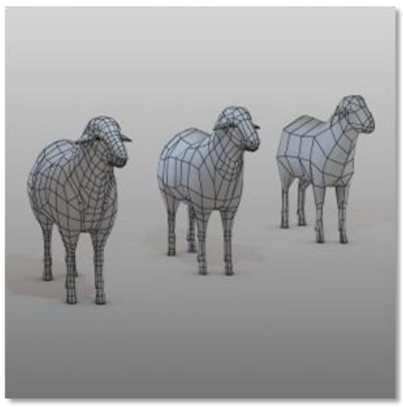

Figure 1. If the user won’t notice the difference, reduce the number of polygons to improve performance

2.2. Understand Your Target Device

Developers should seek to learn as much information about their target platforms as possible in order to understand different architectures. Manufacturers' websites for devices are a good place to look for specifications and they may also provide other helpful developer community resources. The PowerVR Graphics SDK provides architecture documents for reference:

- PowerVR Hardware Architecture Overview for Developers.
- PowerVR Series5 Architecture Guide for Developers.

Even after the graphics architecture is thoroughly understood it is important to remember that other factors such as variations in CPU power, memory bandwidth and thermal load will also impact performance.

2.3. Promote Calculations ‘Up the Chain’

By performing calculations earlier in the pipeline the overall number of operations can be reduced and therefore workload can be substantially reduced. Generally, in a scene, there are far fewer vertices than fragments, therefore processing per-vertex instead of per-fragment would greatly reduce the number of calculations. One use case, for example, could be to perform per-vertex lighting instead of per-pixel lighting.

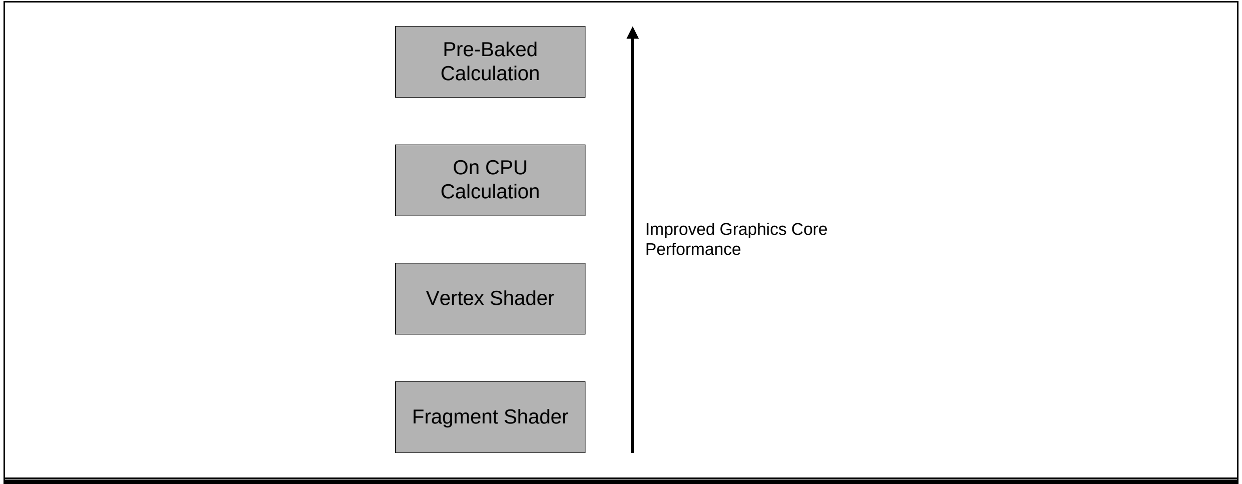

Figure 2. Promote calculations up the chain to reduce the number of times they are executed

It is also possible to consider moving calculations off the Graphics Core altogether. Though the Graphics Core may be able to perform operations far more rapidly than the CPU can, it would be even faster for the CPU to perform an operation just once instead of the allowing the operation to be performed for many vertices on the Graphics Core.

To take the concept even further, consider performing calculations offline, baking values into the scene, effectively replacing expensive run-time calculations with a simple lookup. For example, replacing real-time lighting with lightmaps for static objects in a scene, such as terrain, buildings and trees, can be a particularly effective compromise substantially improving performance and, in many cases, providing higher quality lighting than would be possible to calculate at run-time.

2.4. Use Vertex Buffers and Indexed Geometry

Vertex Buffer Objects (VBOs) enable the graphics driver to cache vertex data attributes, such as texture coordinates (for mapping 2D images to the mesh) and model-space position. For static objects (those whose vertex attributes changes infrequently, if at all), VBOs improve performance as the cached data can be reused to render many frames.

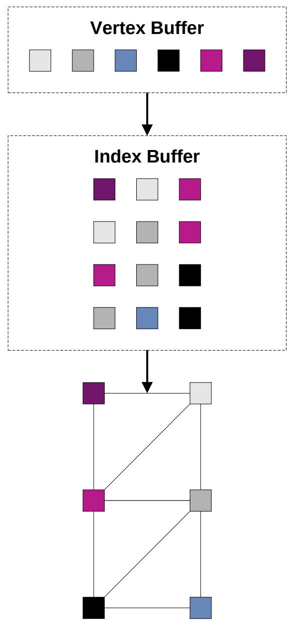

Figure 3. VBOs and IBOs

In the example above, an IBO (Index Buffer Object) is used in conjunction with a VBO. Index buffers define the order in which elements of a vertex buffer should be accessed to represent the triangles in a mesh. IBOs improve performance and reduce the storage space requirements of complex mesh data as vertex attributes are written to the VBO once then referenced as many times as required to represent the triangles surrounding that vertex position.

PowerVR hardware is optimized for indexed triangle lists. For finely-tuned performance, vertex and index buffers should be sorted to improve cache efficiency when the data is accessed by the GPU. Our 3D scene exporter and converter tool, PVRGeoPOD, automatically applies sorting to mesh data when generating POD (PowerVR object data) files.

2.5. Batch Render State

Modifying the GL state machine incurs CPU overhead in the graphics driver, as changes need to be interpreted and converted into tasks that can be issued to the Graphics Core. To reduce this overhead, you must minimise the number of API calls and state changes made by your application.

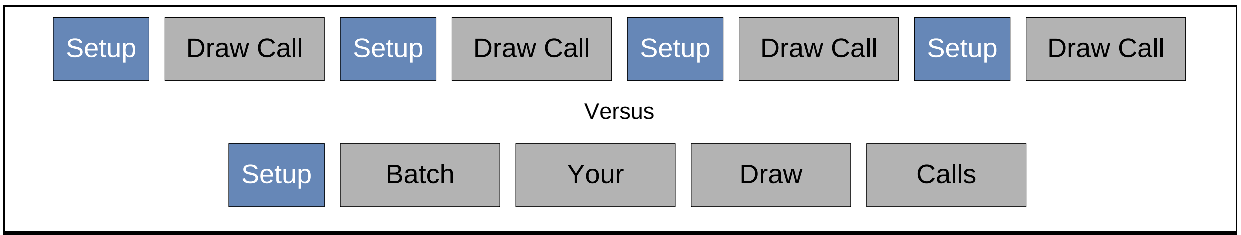

Figure 4. Batch-your-draws!

For geometry data, combine as many meshes into a single draw call as possible. An example use case: Meshes for seats on a train have static position and orientation relative to one another and use the same render state, the seats and the train could all be combined into a single mesh. Now to draw the train interior, several draw calls have merged into a single call.

With the Hidden Surface Removal (HSR) feature on PowerVR hardware it is not necessary to submit geometry in depth order to reduce overdraw. By freeing applications from this restriction they can focus on sorting draws by render state, ensuring state changes are minimised.

Similarly to geometry data, it is possible to combine several textures into a single bindable object by using texture atlases or texture arrays (where available). Textures can then be applied per-object with the appropriate shader uniforms.

As discussed in Avoid Read/Write of In-Use Buffer Objects, modifying buffer data may stall the graphics pipeline or increase the amount of memory allocated by the graphics driver. When batching draws together, it is important to consider the update frequency of buffers. For example, you should batch spatially coherent objects with static vertex data into one VBO, objects with dynamic data (e.g. soft body objects like cloth) into another.

2.6. Use Texture Compression

In some instances it is worth considering the balance between texture size and texture compression. It may be possible to use a larger texture and a low-bitrate compression scheme and achieve a better balance of bandwidth savings and acceptable image quality.

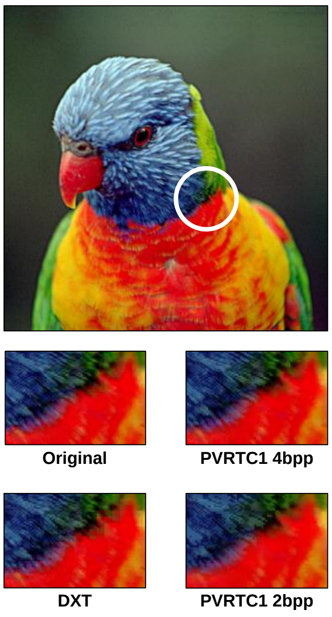

Figure 5. PVRTC texture compression

Texture compression (not to be confused with image file compression) minimises the runtime memory footprint of your textures. This provides several performance benefits; primarily reducing the amount of system memory bandwidth consumed sending data to the Graphics Core. PVRTC and PVRTCII are PowerVR specific compression technologies and will achieve best performance on the hardware, consuming as little as 2 bits per pixel. Depending on the PowerVR generation and graphics API you are targeting, additional compressed texture formats may be supported, for example ASTC.

2.7. Avoid Read/Write of In-Use Buffer Objects

Graphics applications achieve the best performance when the CPU and Graphics Core tasks run in parallel. PowerVR Graphics Cores also operate most efficiently when the vertex processing tasks of one frame are processed in parallel to the fragment colouring tasks of previous frames. When an application issues a command that causes the CPU to interrupt the GPU, it can significantly reduce performance.

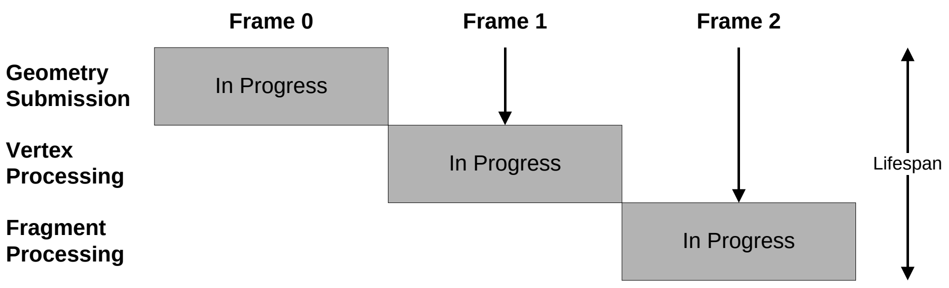

Figure 6. Using the CPU to modify the GPU buffers can be costly

2.7.1. CPU Framebuffer Reads

Accessing the contents of a framebuffer attachment from the CPU is a common cause of slow performance. When such an operation is issued, the calling application's CPU thread must stall until the GPU render has completed. Once the render is complete, the CPU can begin reading data from the attachment. During this time the Graphics Core will not have write access to that attachment which can cause the Graphics Core to stall subsequent renders to that framebuffer. Due to the severe cost these operations should only be used when absolutely necessary, for example to capture a screenshot of a game when a player requests one.

2.7.2. VBO and Texture Modifications

Modifying in-flight VBOs and textures has a cost. As GPUs tend to have at least one frame of latency to ensure they are well occupied with work, altering a buffer required by an outstanding render will give the graphics driver two options:

1. Stall in the buffer modifying API call until the outstanding render completes.
2. Allocate a temporary buffer for the new data so the buffer modifying API call can complete without stalling the CPU thread.

As textures are generally accessed during fragment shading, much later in the graphics pipeline than vertex attributes, the cost of a graphics driver stalling a texture modification is higher than modifying a VBO. Avoiding stalls entirely by creating temporary buffer stores (a.k.a. ghosting) is good for performance, but it may not be desirable for applications that are already running out of buffer storage space.

The stalling and ghosting behaviour of PowerVR varies between GPU and driver versions. For optimal performance, you should only modify VBOs and textures when absolutely necessary. If buffers must be modified, you should use application-side circular buffering so that the GPU can read from one buffer object while the application's CPU thread writes to another.

2.8. Avoid Alpha Test/Discard

When an alpha tested primitive is submitted, early depth testing - such as PowerVR's Hidden Surface Removal (HSR) - can discard fragments that are occluded by fragments closer to the camera. Unlike opaque primitives that would also perform depth writes at this pipeline stage, alpha tested primitives cannot write data to the depth buffer until the fragment shader has executed and fragment visibility is known. These deferred depth writes can impact performance, as subsequent primitives cannot be processed until the depth buffers are updated with the alpha tested primitive's values.

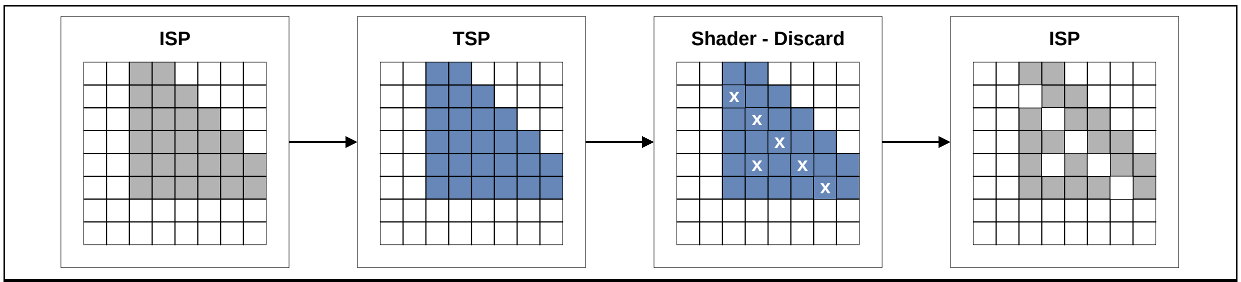

Figure 7. PowerVR alpha test/discard pipeline

For optimal performance, consider alpha blending instead of alpha test to avoid costly deferred depth write operations. To ensure HSR removes as much overdraw as possible, you should also submit your draws in the following order: opaque, alpha-tested, blended.

2.9. Prevent Redundant Framebuffer Memory Transfers

System memory bandwidth accesses use more power than any other GPU operation. Keeping memory accesses to a minimum will reduce the chances of your application being memory bandwidth bound and will also reduce the power consumption of your app.

On tile based architectures, OpenGL ES requires the previous framebuffer attachments to be uploaded to each tile at the start of a render and all framebuffer attachment data to be written to system memory at the end of the render. In many cases these memory transfer operations are redundant. Most applications need to generate a colour image at the end of the render, but have no need to preserve depth and stencil data between frames. Even fewer applications have a genuine need to upload the contents of the colour buffer's previous contents at the start of a new frame.

If the data previously written to framebuffer attachments is not required, you can call `glClear` at the start of the render to prevent the data being uploaded to the tiles. The `glDiscardFramebufferEXT` or `glInvalidateFramebuffer` commands enable your application to inform the graphics driver when buffers should not be written from tile memory to system memory, for example depth and stencil data.

3. Contact Details

For further support, visit our forum:
http://forum.imgtec.com

Or file a ticket in our support system:
https://pvrsupport.imgtec.com

To learn more about our PowerVR Graphics SDK and Insider programme, please visit:
http://www.powervrinsider.com

For general enquiries, please visit our website:
http://imgtec.com/corporate/contactus.asp