PowerVR Hardware

Architecture Overview for Developers

Filename	:	PowerVR Hardware.Architecture Overview for Developers
Version	:	PowerVR SDK REL_18.1@5080009 External Issue
Issue Date	:	31 May 2018
Author	:	Imagination Technologies Limited

Contents

List of Figures

1. Introduction

The purpose of this document is to provide developers with an overview of the PowerVR graphics hardware architecture. The PowerVR architecture is based on a concept called Tile Based Deferred Rendering, commonly shortened to TBDR. TBDR focuses on removing redundant operations as early as possible in the graphics pipeline, minimising memory bandwidth use and power consumption while improving processing throughput.

Note: This document assumes you are familiar with the 3D graphics programming pipeline of OpenGL, DirectX or a similar 3D graphics programming API.

2. Overview of Modern 3D Graphics Architectures

As shown in Figure 1, a modern System on Chip (SoC) often integrates both a CPU and a Graphics Processor. The CPU is optimised for processing sequential, heavily branched data sets that require low memory latency, dedicating transistors to flow control and data caches.

The graphics core, on the other hand, is optimised for repetitive processing of large, unbranched data sets, such as in 3D rendering. Transistors are dedicated to registers and arithmetic logic units rather than data caches and flow control.

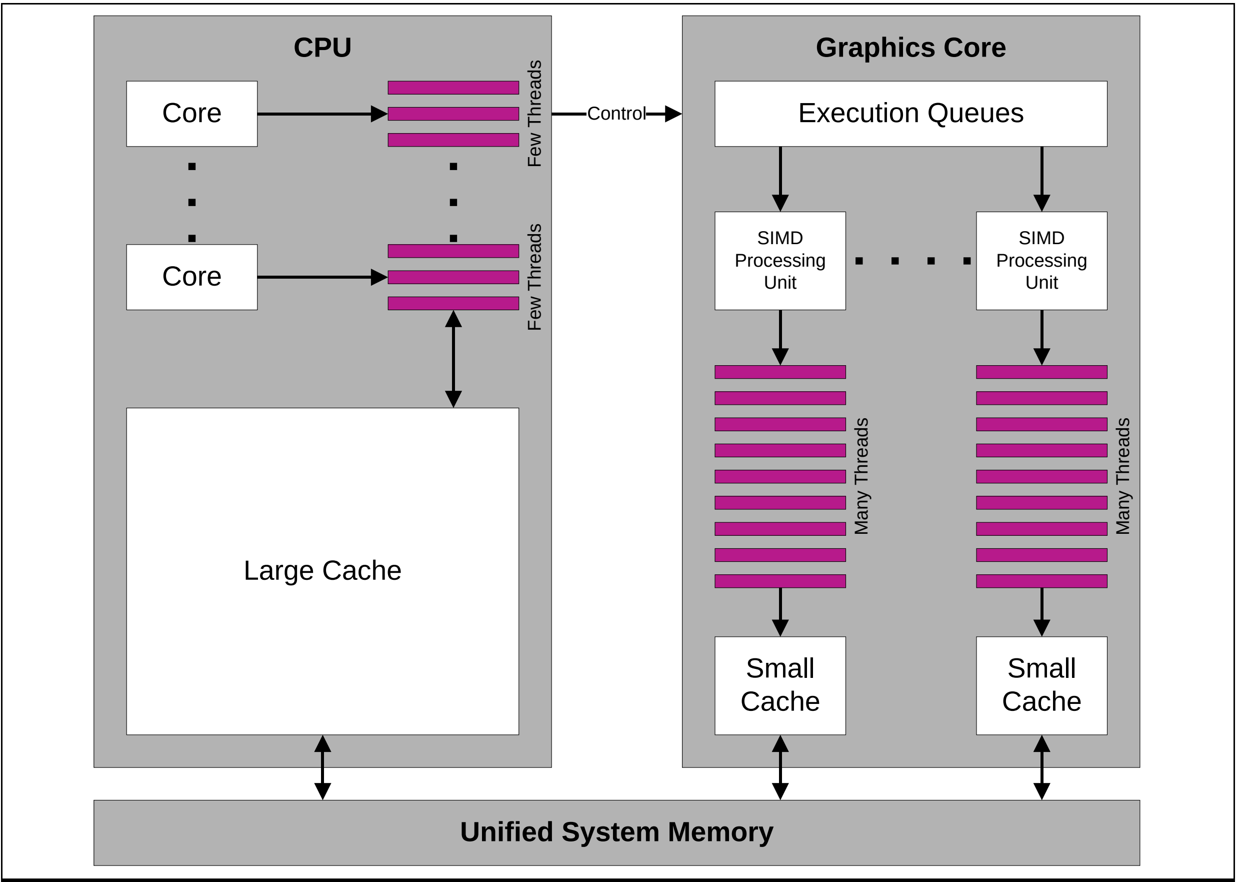

Figure 1. SoC architecture overview

2.1. Single Instruction, Multiple Data

Typical CPUs are optimised to execute large, heavily branched tasks on a few pieces of data at a time. A thread running on a CPU is often unique and is executed on its own, largely independent of all other threads. Any given processing element will process in just a single thread. Typical numbers of threads for a program on a CPU is commonly one to eight, up to a few tens at any period of time.

Graphics processors work on the principle that the exact same piece of code will be executed in multiple threads, often numbering into the millions to handle the large screen resolutions of today's devices. These threads differ only in input and normally follow the exact same execution steps.

To do this efficiently, each graphics processor executes the same instruction on multiple threads concurrently, in a form of Single Instruction, Multiple Data (SIMD) processing. SIMD processors are typically either scalar, which means operating on one element at a time, or vector, which means operating on multiple elements at a time.

2.1.1. Parallelism

The main advantage of the SIMD architecture is that significant numbers of threads can be run in parallel for a correctly structured application and this is done with extremely high efficiency. SIMD architectures are usually capable of running many orders of magnitude more threads at once than a typical CPU. SIMD is designed to operate on large coherent data sets and performs exceptionally well at this type of task. Algorithms that operate independently on a large coherent data set, such as graphics and image processing, are well suited for this processor type.

2.2. Vector and Scalar Processing

Modern graphics core architectures feature multiple processing units which are either vector or scalar based. Both are supported by different versions of PowerVR architecture – Series 5 supporting Vector, and Series 6, Series 7 and Series 8 supporting Scalar.

Scalar processing units operate on a single value per processing unit. Vector processing units work on multiple values per processing unit.

2.2.1. Vector

Vector processing can be very efficient, as the execution unit can work on multiple values at the same time rather than just one. For colour and vertex manipulation, this type of architecture is extremely efficient. Traditional rendering operations are, therefore, well suited to this architecture as calculations often operate on 3 or 4 elements at once.

The main drawback of vector architectures is that if scalar values or vectors smaller than the processor expects are used, the additional processing element width is wasted. The most common vector width is 4, which means that a shader or kernel mainly operating on 3 component vectors will operate these instructions with 75% efficiency. Having a shader that works on only one scalar at a time may take this number down to as low as 25%. This wastes energy and performance as parts of the processor are not doing any work. It is possible to optimise for this by vectorising code, but this introduces additional programmer burden.

2.2.2. Scalar

Scalar processors tend to be more flexible in terms of the operations that can be performed per hardware cycle, as there is no need to fill the additional processing width with data. Whilst vector architectures could potentially work on more values in the same silicon area, the actual number of useful results per clock will usually be higher in scalar architectures for non-vectorised code. Scalar architectures tend to be better suited to general purpose processing and more advanced rendering techniques.

3. Overview of Graphics Architectures

Modern graphics architectures can be classified by the following types:

- Immediate Mode Renderer (IMR)
- Tile Based Renderer (TBR)
- Tile Based Deferred Renderer (TBDR)

Additionally, the shading architecture can be unified or non-unified. This section of the document explains the key differences between these graphics architectures.

3.1. Unified Architecture and Non-Unified Architectures

Unified shader architecture executes shader programs, such as fragment and vertex shaders on the same processing modules. A non-unified architecture uses separate dedicated processing modules for vertex and fragment processing.

Figure 2 shows how a unified architecture can save power and increase performance compared to a non-unified architecture. Unified architectures also scale much more easily to a given application, whether it is fragment or vertex shader bound, as the unified processors will be used accordingly. All PowerVR hardware platforms have unified shader architecture.

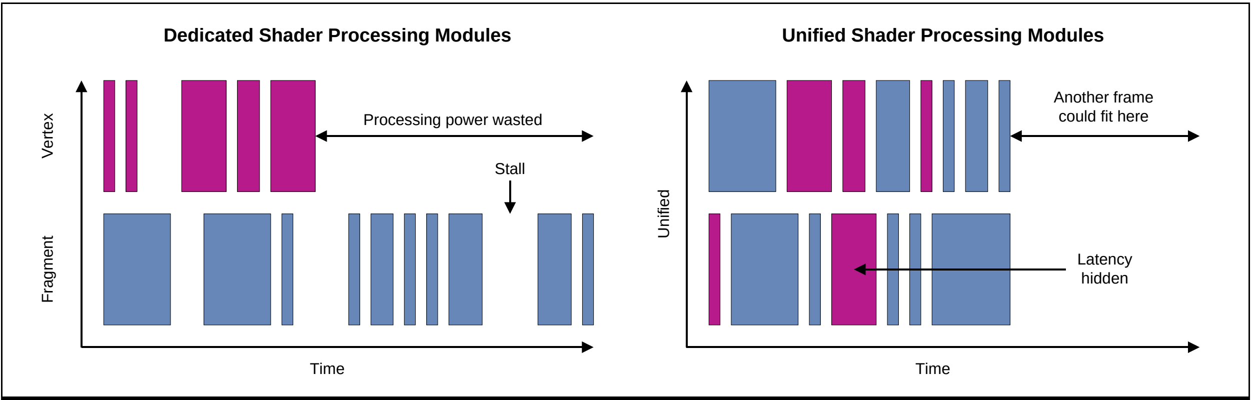

Figure 2. Non-unified vs unified architecture utilisation

3.2. Overdraw

The term "overdraw" refers to wastefully colouring pixels that do not contribute to the final image colour. Overdraw occurs when the pixels coloured for a drawn object are overwritten by another object. Most applications submit draws as triangle meshes that can render in front of each other or even intersect, making overdraw inevitable. To keep overdraw to a minimum, graphics cores incorporate overdraw reduction techniques such as Early-Z testing. The efficiency of these techniques can be very dependent on an application's draw call submission order.

3.3. Common Architectures

3.3.1. Immediate Mode Rendering (IMR)

In a traditional Immediate Mode Rendering (IMR) architecture, each submitted object travels through the entire pipeline immediately, being transformed, rasterized and coloured before the next object is processed. There are a number of inefficiencies associated with a simple IMR architecture that lead to wasted processing power and memory bandwidth. Figure 3 identifies a typical IMR rendering pipeline.

Most modern IMR architectures utilise Early-Z techniques to perform depth tests early in the graphics pipeline, reducing the amount of overdraw in a render (see Figure 3). Applications can only fully benefit from this optimisation if geometry is always submitted to the hardware in front to back order, which requires per frame sorting for scenes with moving cameras and/or geometry.

As IMRs store all colour, depth and stencil buffers in system memory, regular Read-Modify-Write operations to these buffers can quickly induce a large system memory bandwidth overhead. A modern IMR will have a large graphics cache aiming to reduce system memory usage.

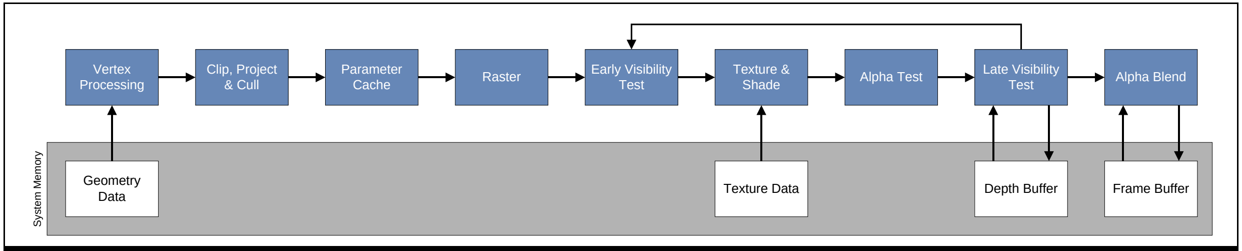

Figure 3. IMR pipeline

3.3.2. Tile Based Rendering (TBR)

Tiling is the process of binning post-transform geometry data into small rectangular regions, called tiles. Rasterization and fragment processing then occurs on a per-tile basis. Figure 4 depicts the Tile Based Rendering (TBR) pipeline. TBR consists of two phases namely vertex processing and per-tile rasterization.

By processing a tile at a time, the size of on-chip buffers can be finely tuned to the tile size. The graphics hardware can then use on-chip buffers for colour, depth and stencil buffer Read-Modify-Write operations. This enables the hardware to avoid costly system memory transfer operations and, instead, use high speed on-chip memory.

Although the TBR approach improves on the traditional IMR design, it does not attempt to reduce overdraw. When rendering each tile, geometry is processed in submission order. Obscured fragments will still be processed, resulting in redundant colour calculations and texture data fetches. Early-Z techniques can be used to reduce overdraw. As with an IMR, applications must sort and submit geometry from front to back to maximise the benefit of Early-Z overdraw reduction.

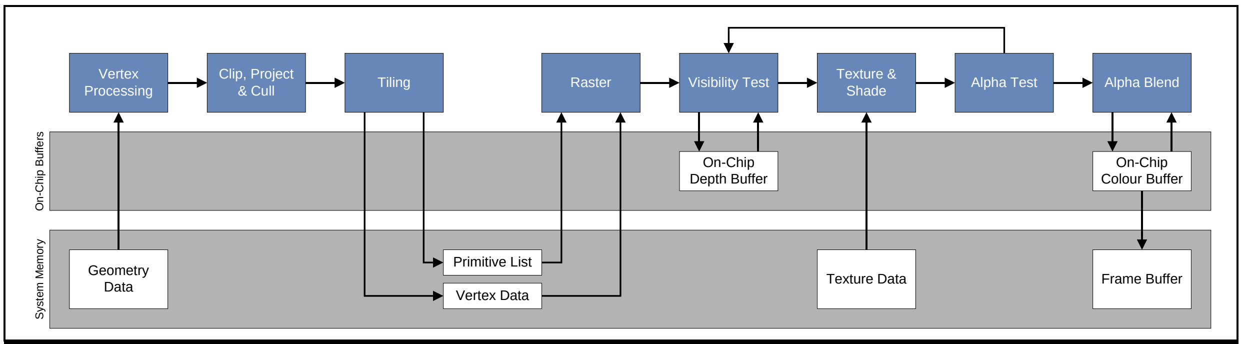

Figure 4. TBR pipeline

3.3.3. Tile Based Deferred Rendering (TBDR)

Figure 5 illustrates Tile Based Deferred Rendering (TBDR) pipeline. TBDR rendering splits the per-tile rendering process into two stages namely Hidden Surface Removal (HSR) and deferred pixel shading.

When a scene composed of three-dimensional objects is created, some of the objects and surfaces may obscure all or parts of others. Hidden Surface Removal is the process by which the obscured sections of objects in a scene are removed from the render.

Deferred rendering means that the architecture will defer all texturing and shading operations until all objects that could be deferred, primarily opaque geometry, have been tested for visibility. The efficiency of HSR is such that overdraw can be removed entirely for completely opaque renders. This significantly reduces system memory bandwidth requirements.

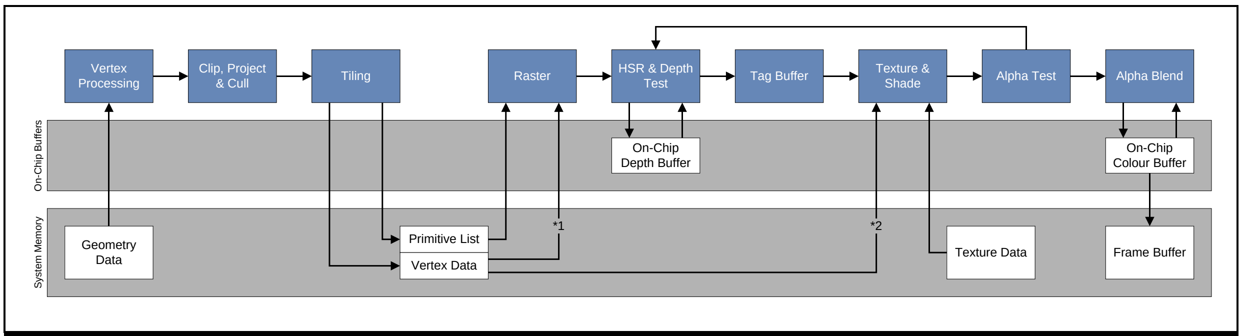

Figure 5. TBDR pipeline

4. What is PowerVR?

PowerVR is the name of the graphics hardware IP family from Imagination Technologies. All generations are based on our patented Tile Based Deferred Rendering (TBDR) architecture. The core design principle of the TBDR architecture is to keep the system memory bandwidth requirements of the graphics hardware to a bare minimum.

As data transferred to and from system memory is the biggest cause of graphics hardware power consumption, any reduction made in this area will allow the hardware to operate at a lower power. Additionally, the reduction in system memory bandwidth use and the hardware optimisations associated with it, such as using on-chip buffers, enables an application to execute its render at a higher performance than other graphics architectures.

Due to the balance of low-power and high-performance, PowerVR graphics cores are dominant in the mobile and embedded devices market.

4.1. PowerVR Architecture Overview

4.1.1. Vertex Processing (Tiler)

Each frame, the hardware processes submitted geometry by executing application-defined transformations, such as vertex shaders, and then converting the resultant data to screen-space. The Tile Accelerator (TA) then determines which tiles contain each transformed primitive. Once this is known, per-tile lists are updated to track the primitives which fall within the bounds of each tile. The transformed geometry and tile lists are both stored in an intermediate store called the Parameter Buffer (PB). This store resides in system memory and contains all information needed to render the tiles. Figure 6 provides an example of a frame buffer divided into tiled regions.

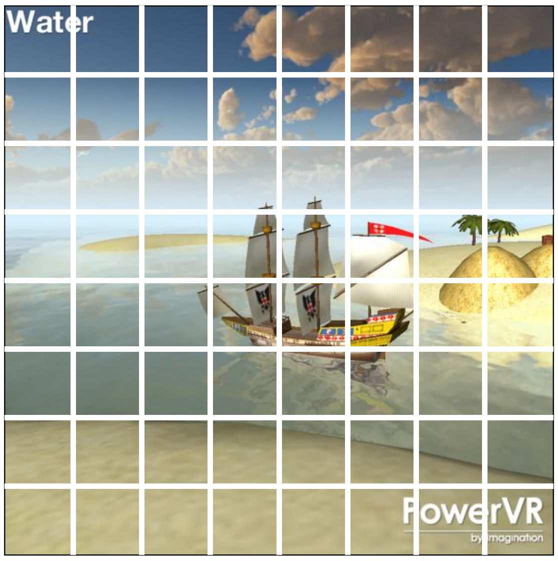

Figure 6. An example of a frame buffer divided into tiled regions

4.1.2. Per-Tile Rasterization (Renderer)

Rasterization and pixel colouring is done on a per-tile basis. When a tile operation begins, the corresponding tile list is retrieved from the PB to identify the screen-space primitive data that needs to be fetched. The Image Synthesis Processor (ISP) fetches the primitive data and performs Hidden Surface Removal (HSR), along with depth and stencil tests. The ISP only fetches screen-space position data for the geometry within the tile.

This is followed by the Texture and Shading Processor (TSP), which applies colouring operations, like fragment shaders, to the visible pixels. Once a tile’s render is complete, the colour data is written to the framebuffer in system memory. This process is repeated until all tiles have been processed and the frame buffer is complete.

4.1.3. On-Chip Buffers

Read-Modify-Write operations for the colour, depth and stencil buffers are performed using fast on-chip memory instead of relying on repeated system memory access, as traditional IMRs do. Attachments that the application has chosen to preserve, such as the colour buffer, will be written to system memory.

4.1.4. PowerVR Shader Engine

The PowerVR shader engine is based on a massively multi-threaded and multi-tasking approach. It is hardware managed and load balanced by using a data driven execution model to ensure the highest possible utilisation efficiency. This approach schedules tasks based on data availability and enables switching between independent processing tasks to ensure that data dependency stalls are avoided at all costs.

4.1.5. Firmware

In many graphics architectures, hardware graphics events are handled on the CPU by the graphics driver. All PowerVR graphics cores are managed by firmware, enabling the graphics processor to handle the majority of high level graphics events internally. This approach keeps event handling latency to a minimum and reduces the graphics driver's CPU overhead.

4.2. Hidden Surface Removal Efficiency

Figure 7 demonstrates

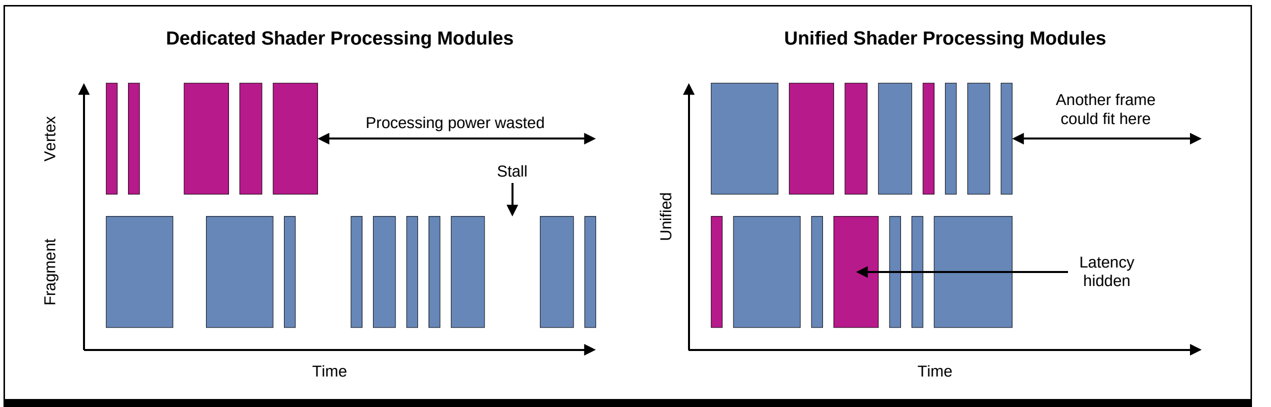
Dedicated Shader Processing Modules
Vertex
Fragment
Processing power wasted
Stall
Time
Unified Shader Processing Modules
Unified
Another frame could fit here
Latency hidden
Time

Figure 2. Non-unified vs unified architecture utilisation

Overdraw. In a traditional IMR architecture, this scene would cause green and red colours to be calculated for the sphere and cube respectively in the areas that are occluded by the yellow cone. In architectures that include Early-Z testing, an application can avoid some overdraw by submitting draw calls from front to back. Submitting in this order builds up the depth buffer so occluded fragments further from the camera can be rejected early. This creates additional burden for the application, as draws have to be sorted every time the camera or objects within the scene move. Additionally, it doesn't remove all overdraw as sorting per-draw is very coarse. For example, it doesn't resolve overdraw caused by object intersection. It also prevents the application from sorting draw calls to keep graphics API state changes to a minimum.

In the PowerVR TBDR, Hidden Surface Removal (HSR) will completely remove overdraw regardless of draw call submission order.

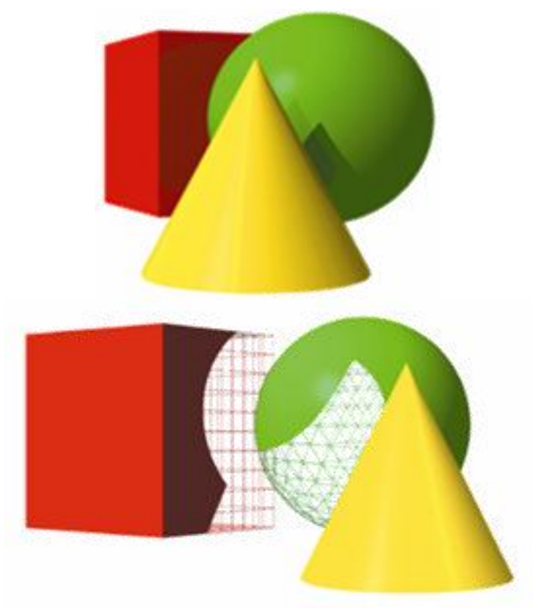

Figure 7. Example of overdraw with opaque objects

Figure 8 is a screen capture from MadFinger Game's Shadowgun. Figure 9 highlights the amount of overdraw in the same scene, ignoring Early-Z or HSR optimisations that may be applied by a graphics core. The closer to white a pixel is, the more overdraw is present.

In this frame, 4.7 fragments are coloured on average per screen pixel. On a PowerVR device, 1.2 fragments are coloured on average per screen pixel, which is 75% fewer fragments than the application submitted. Figure 10 shows the amount of "PowerVR overdraw" (post-HSR) for the same captured frame. The render doesn't achieve a 1:1 ratio between coloured fragments per screen pixel as the scene isn't completely opaque, because blended UI elements are contributing to the average.

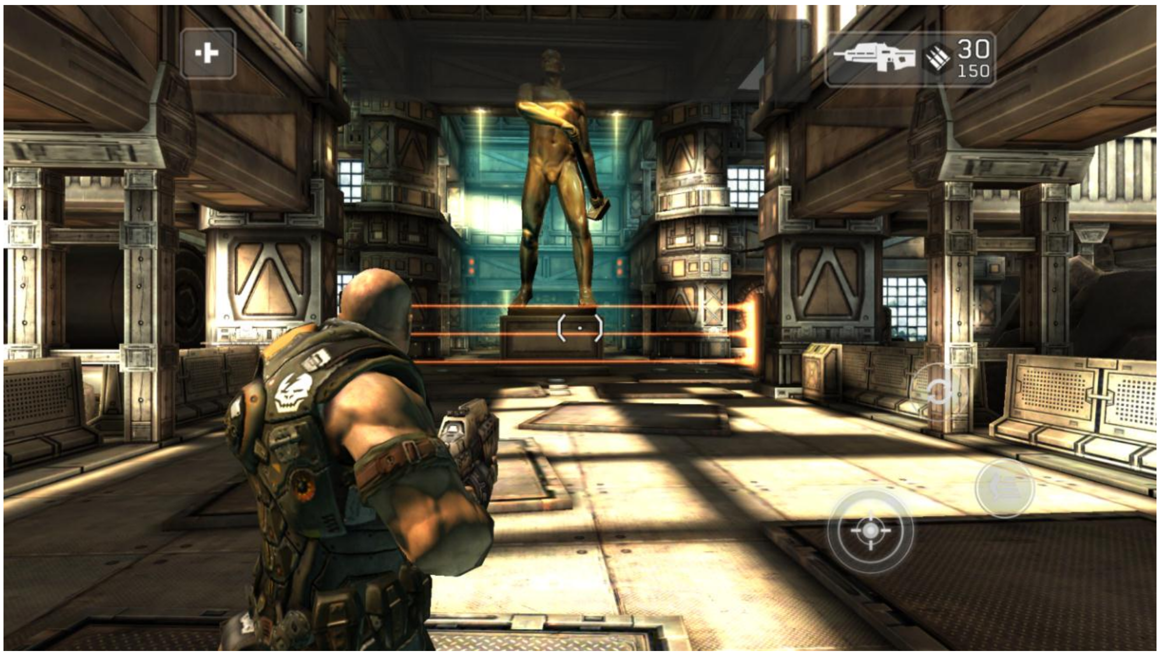

Figure 8. Shadowgun original frame

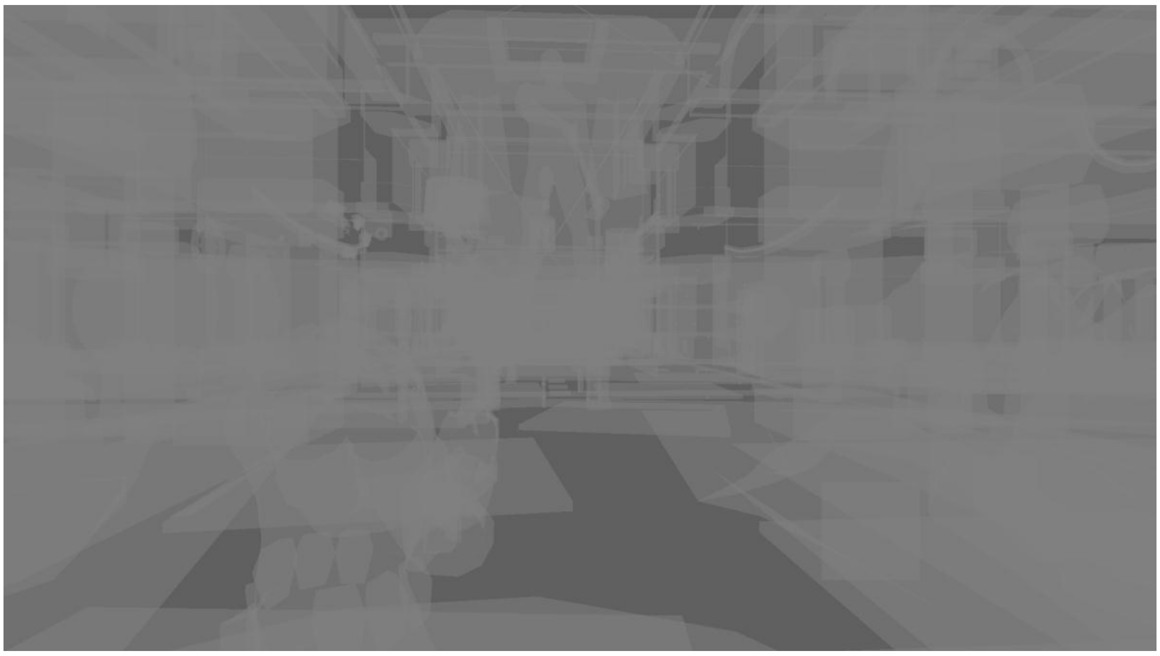
Figure 9. Shadowgun overdraw

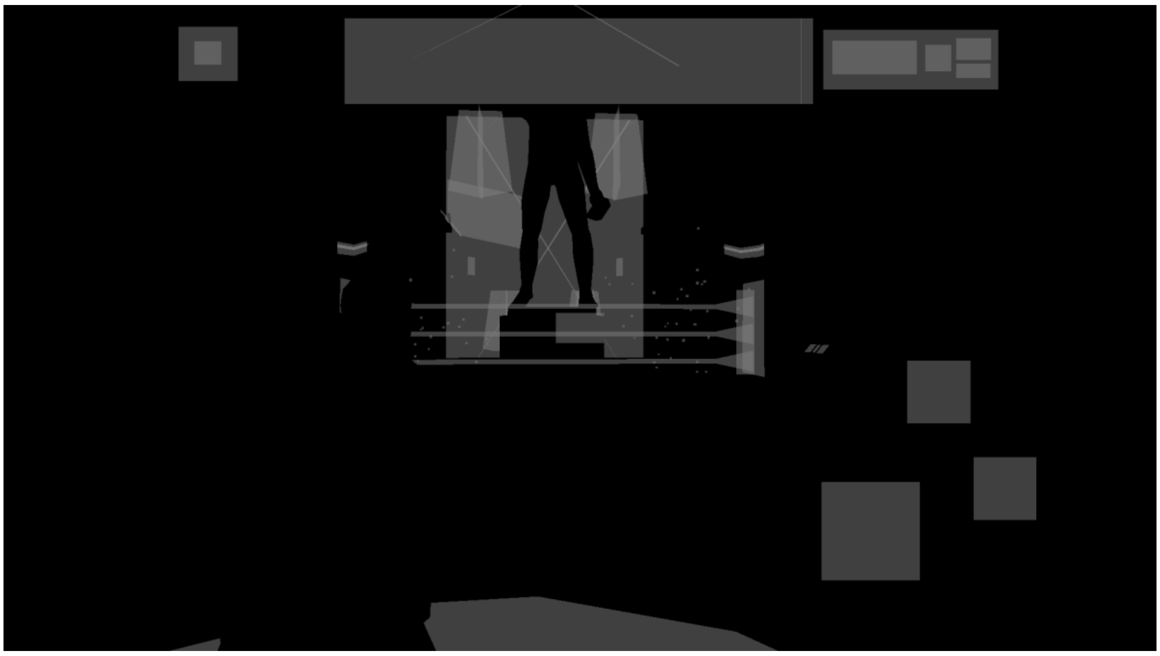
Figure 10. Shadowgun PowerVR overdraw

5. Further Information

Over the years, there have been many generations of the PowerVR hardware family. All modern PowerVR generations are based on the Tile Based Deferred Rendering architecture outlined in Section 3.3.3. These generations (Series5, Series5XT, Series6, Series6XT, Series 7 and Series 8XE) are commercially available and actively targeted by 3D graphics developers.

For more information regarding the PowerVR hardware family, refer to the Imagination website:

http://www.imgtec.com/powervr/

For more detailed information regarding the PowerVR hardware architecture, contact us.

6. 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

Appendix A. Glossary

Term	Meaning
ALU	Arithmetic Logic Unit. It is responsible for processing shader instructions.
Early-Z	An umbrella term for a collection of optimisations commonly used by graphics cores. Early-Z techniques reduce overdraw by performing depth tests early in the graphics pipeline.
firmware	A dedicated program running on the graphics core that handles hardware events (for example, a tile processing operation completing).
fragment	The data necessary to calculate a pixel colour. Multiple fragments may contribute to the colour of a pixel (for example, when a transparent object is drawn in front of an opaque object).
graphics pipeline	The sequence of processing stages within a graphics core that must be executed to render an image.
HSR	Hidden Surface Removal.
IMR	Immediate Mode Renderer.
ISP	Image Synthesis Processor.
overdraw	The term “overdraw” refers to wastefully colouring pixels that do not contribute to the final image colour.
SIMD	Single Instruction, Multiple Data. Concurrent execution of a single instruction across multiple ALUs, where each ALU has unique input and output.
scalar [shader architecture]	A shader architecture in which an ALU processes a single value at a time.
pixel	The smallest addressable area of a framebuffer.
rasterization	The process of determining which pixels a given primitive touches.
render	The process of converting application submitted data into coloured pixels that can be stored in the framebuffer.
renderer	The tile processing stage of a TBDR pipeline. This includes rasterization and fragment shading.
TA	Tile Accelerator.
TBR	Tile Based Renderer.
TBDR	Tile Based Deferred Renderer.
tile	A rectangular group of pixels. In TBR and TBDR architectures, the framebuffer is broken into many tiles. The tile size of each PowerVR graphics core is decided during hardware design, typically 32x32 pixels.
tiler	The vertex shading, clipping, projection, culling and tiling stages of a TBDR pipeline.
TSP	Texture and Shading Processor.

Term	Meaning
vector [shader architecture]	A shader architecture in which an ALU processes multiple values simultaneously. Vector architectures commonly have a width of 4, allowing the ALU to calculate values for the ‘x’, ‘y’, ‘z’ and ‘w’ components of a vector data type.