數位電路實驗

Final Project – Real-time Rendering

B04901060 黃文璁、B04901080 戴靖軒、B04901048 陳則宇

目錄

一、動機

為什麼會想做渲染呢？其實原因很簡單：

（一） 喜歡玩遊戲

遊戲目前是 real-time rendering 的主要應用範圍之一，相信多數人都有玩遊戲的經驗，而我們一向覺得遊戲能將場景、角色模型顯示在螢幕上是很奇妙的事，故我們決定朝這個方向訂出我們的專題題目，好學習這方面的知識。

（二） 想了解渲染基本原理

除了玩遊戲外，我們也曾經利用 Unity 或 Unreal 等遊戲引擎製作遊戲，遊戲引擎基本上都會幫我們處理許多 rendering 的實作細節，但我們想了解其中原理並試著實作簡單的 renderer，有機會的話再實作遊戲引擎的其他部分。

（三） 看到之前數電實驗有人做過 3D 效果

先前有人做過 3D 視覺效果，但做法都是採用 2D 模擬 3D，而不是真正渲染。

舉例來說：

- [2014 Fall] 互動式水族箱[1]

  這組使用 24 張不用角度的鯉魚王圖片來模擬 3D 效果。

- [2009 Spring] 魔術方塊[2]

  這組先將魔術方塊旋轉過程在電腦上渲染好，再將圖片存至 FPGA 上。

- [2006 Fall] GeniusFarNchia 第一人稱賽車遊戲[3]

  這組使用 2D 圖片的透視效果，模擬 3D 場景。

基本上都是限制較多的 3D 應用，我們希望能藉由直接執行渲染來增加通用性。

（四） 想了解 GPU 架構

近幾年來 GPU 在圖形以外的應用領域，也起了很大作用。若能夠藉由這次機會學習到 GPU 的架構和實作方式，也是我們所期待的。雖然最後因為時間問題還來不及實作出來，不過若有機會的話，希望接下來幾屆有修這門實驗的同學可以試著實作看看。

---

[1] https://www.youtube.com/watch?v=yhH2YFyFTjk
[2] https://www.youtube.com/watch?v=QLDrR8_E4T4
[3] https://www.youtube.com/watch?v=UGQ8hbWYLhY

二、目的

以下是這次期末專題所要完成的功能：

（一）讓 FPGA 得以執行 C/C++ 程式。

在前三個實驗中，我們撰寫的是 Verilog，用來描述硬體。但由於期末專題希望能做出渲染、檔案讀取等較複雜的功能，故若可以用軟體編寫能大幅降低開發難度，雖然如此一來硬體的優勢也跟著減少，除非再撰寫其他加速用的硬體。

（二）讓軟體得以從外接儲存裝置讀取檔案。

對於 DE2-115 來說，外接儲存裝置的主要選項有 USB Mass Storage 和 SD 卡兩種，由於 USB 的 Protocol 相對來說非常複雜，故我們選用 SD 卡來儲存模型和貼圖。

（三）可以 parse 讀取的模型、貼圖檔案。

在眾多模型檔案格式中，我們選用 Wavefront .obj 格式。這是由於.obj 格式是純文字檔且語法簡單，可以直接閱讀，也相對容易 Parse。至於貼圖檔案則選擇 PNG。

（四）可以執行基於柵格化 (rasterization) 的渲染。

對目前多數有 real-time 需求的渲染，主要是基於 rasterization 而不是物理上更精確的 ray-tracing 等方法，這是由於 rasterization 在計算和執行上更簡單。

（五）支援貼圖、頂點法向量和平滑渲染 (smooth shading)。

對於有提供貼圖和頂點貼圖座標的模型，我們可以進行 texture mapping 來為模型加上貼圖。而對於提供頂點法向量的模型則可以進行 smooth shading 來平滑化模型的光影。

（六）讓渲染速度達到 real-time。

針對這次專題提升 framerate 的手段可以後文七、效能比較。

（七）可以使用滑鼠、鍵盤操作

如同一般遊戲用 WASD 平移鏡頭，另外還有 QE 上下平移鏡頭和 IJKL 旋轉鏡頭。滑鼠則用來使鏡頭圍繞 Y 軸旋轉，這部分可參考使用說明書。

以下是這次期末專題希望達成但尚未實作的功能：

（一）實作類似 GPU 的硬體

由於 FPGA 的優勢還是在硬體，故我們原先希望能夠實作一個額外的圖形處理硬體來和 Nios II 合作，發揮 FPGA 的硬體優勢，不過由於實作的難度較高以及背景知識的不足，目前還無法實作出來，希望將來有機會能繼續了解這方面的知識。

（二）實作遊戲

實作遊戲則是需要一個完整的遊戲引擎，包含渲染器、物理引擎、Entity-component-system (ECS) 等部分，在軟體上的複雜性很高，且由於我們沒有實作 GPU，所以在 framerate 和解析度上仍不夠理想，故這部分也尚未實作出來。

三、原理

（一） Qsys

Qsys 提供 GUI 讓我們可以免去在 Verilog 裡面接線的困擾，更重要的是他可以使用 Altera 提供的許多 IP 免去自行撰寫的麻煩，此外還可以基於 Avalon-MM 很簡單的建立 multi-master bus 系統，Qsys 會自動幫我們生成硬體，大幅降低多 master 存取資源的處理麻煩。最後這個工具可以幫我們合成出 Verilog 或 VHDL 硬體描述，再藉由 Quartus 合成電路，即可在 FPGA 上使用。

關於 Qsys 的其他詳細說明可以參考官方文件[4]。

（二） Nios II、Nios II HAL

Nios II 是一個處理器，特點是可以合成到 FPGA 上，以及可以自訂指令集，增加開發的彈性。有了 Nios II 和 Quartus 隨附的 Eclipse + Nios II 開發環境以後，我們得以在 FPGA 上開發 C/C++ 程式，增加了 FPGA 的應用範圍，我們得以同時開發軟硬體。

另外根據官方文件[5]的描述，Nios II 在 Cyclone IV 上可以達到 160MHz 上限，故我們可以利用 PLL 提高 clock speed 來增加他的效能。

至於在 Nios II 的軟體方面勢必有需要和硬體進行溝通，Nios II 使用了 HAL (Hardware Abstraction Layer) 的概念，將硬體操作抽象化，這部分描述可以參考官方對於 Nios II 軟體開發的說明文件[6]。在本次實驗中我們主要使用到 HAL I/O 的部分，也就是透過 Avalon-MM 讀寫記憶體位置，I/O 介面是以 C Macro 的形式使用，介面如下：

Macro	Use
`IORD(BASE, REGNUM)`	Read the value of the register at offset REGNUM in a device with base address BASE. Registers are assumed to be offset by the address width of the bus.
`IOWR(BASE, REGNUM, DATA)`	Write the value DATA to the register at offset REGNUM in a device with base address BASE. Registers are assumed to be offset by the address width of the bus.
`IORD_32DIRECT(BASE, OFFSET)`	Make a 32-bit read access at the location with address BASE+OFFSET.
`IORD_16DIRECT(BASE, OFFSET)`	Make a 16-bit read access at the location with address BASE+OFFSET.
`IORD_8DIRECT(BASE, OFFSET)`	Make an 8-bit read access at the location with address BASE+OFFSET.
`IOWR_32DIRECT(BASE, OFFSET, DATA)`	Make a 32-bit write access to write the value DATA at the location with address BASE+OFFSET.
`IOWR_16DIRECT(BASE, OFFSET, DATA)`	Make a 16-bit write access to write the value DATA at the location with address BASE+OFFSET.
`IOWR_8DIRECT(BASE, OFFSET, DATA)`	Make an 8-bit write access to write the value DATA at the location with address BASE+OFFSET.

[4] https://www.altera.com/products/design-software/fpga-design/quartus-prime/features/qts-qsys.html
[5] https://www.altera.com/en_US/pdfs/literature/ds/ds_nios2_perf.pdf
[6] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf

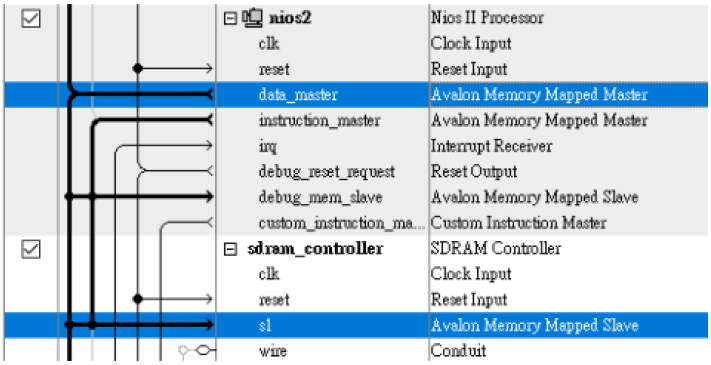
（三） Floating-point Hardware 2 (FPH2)

Nios II 是個能自訂指令集的處理器，FPH2 則是 Altera 提供的 Nios II 浮點運算指令集[7]，使用 FPH2 可以大幅提升浮點運算的效能，而由於渲染的運算中有大量浮點運算，故加上 FPH2 帶來的效益很大，實際的效能差異可以參考七、效能比較。

（四） Avalon Memory-mapped Interface (Avalon-MM)[8]

Avalon-MM 介面讓我們得以實現 multiple-master multiple-slave 的架構，在 Qsys 中連接各個 Avalon-MM 介面後，軟體可以幫我們產生出複雜的 interconnect 架構，而不用自行處理如 arbitration 等功能。對本專題來說有 SRAM 需要同時被 Nios II 和 Frame reader 讀取，所以若不使用 Avalon-MM 的話就需要自行處理多個 master 產生的問題。

更重要的是，Avalon-MM 雖然功能強大，但在 Qsys 中看起來不過就是把線接起來而已，使用上也非常容易。下圖利用 Avalon-MM 連接 Nios II 和 SDRAM Controller：



（五） Avalon Streaming Interface (Avalon-ST)

Avalon-ST 一般用來處理單向的資料流，常見的如影像、音訊等，在本次專題中主要用於處理 Frame 資料讀出成影像後的後續處理，一直到最後輸出至螢幕為止。

（六） Avalon Interrupt Interface

有些事件會隨時觸發，需要立刻處理，要跳出程式一般的執行步驟。這個部分可以藉由 Avalon Interrupt Interface 的 IRQ (Interrupt Request) 功能來實現，支援 IRQ 的模組可以在事件發生時產生 IRQ 訊號，傳送給 IRQ 接受者如 Nios II。在 C/C++ 程式中我們可以利用 “sys/alt_irq.h” 中提供的函數 alt_irq_register() 來進行註冊，決定需要觸發的函數，這部分將在六、軟體架構中描述。

---

[7] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_fph2.pdf
[8] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf

（七） PS/2 滑鼠、鍵盤

PS/2 本身的協定和 I^2C 有些類似，可以參考文件說明[9]。

至於鍵盤和滑鼠的封包內容，這次實驗主要用到下列部分：

鍵盤： 按下：收到 keycode

放開：收到 240 後，再收到 keycode

滑鼠： 滑鼠有動作時，會收到 3 個 byte。如下圖

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X movement							
Byte 3	Y movement							

另外如先前提到的，滑鼠、鍵盤的事件要隨時監聽，這部分可以用 IRQ 功能實現，詳細見後文六、軟體架構。

（八） 其他 Altera 提供的資源

1. Altera University Program (UP)[10]

這是 Altera 針對大學方面提供的一些教材和 IP，在這次專題中我們使用到 UP 提供的 SD Controller，他提供了對 FAT16 SD 卡的支援，讓我們得以在 C 裡面支援檔案系統，可以像一般程式一樣根據檔案路徑讀取檔案，大幅降低檔案讀取的難度。

2. SDRAM Controller[11]

在 Qsys 中預設就可以找到 SDRAM Controller，這大概也是因為 SDRAM 本身在使用上並不是很容易，所以若沒辦法完全理解 datasheet[12]的內容的話，不如就暫時使用這個 SDRAM Controller 吧。值得注意的是這個 Controller 需要透過 Avalon-MM 介面來讀寫 SDRAM，這部分在前三個實驗中應該已經能了解操作方式。

3. Altera Video and Image Processing Suite (Altera VIP)

這部分則是 Altera 所提供的一系列針對影像處理的模組，其中對我們來說最重要的是「Frame Reader」模組，他的作用是利用 Avalon-MM 讀取記憶體中的 Frame 然後轉換成 Avalon-ST 格式，可以提供 Altera VIP 的其他模組使用，擔任了兩種 Avalon 介面的橋樑。至於其他用到的模組可以參考後文五、硬體架構。

---

[9] http://www.computer-engineering.org/ps2protocol/

[10] https://www.altera.com/support/training/university/materials-ip-cores.html

[11] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf

[12] http://www.issi.com/WW/pdf/42S16320B-86400B.pdf

四、渲染原理

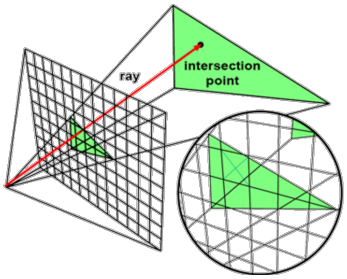
（一）概念

所謂的渲染 (Rendering)，就是將一個由檔案或數學模型描述的場景，根據設定的光線、投影方式，投影到「鏡頭」上，最後產生 2D 的影像。這其中牽涉的數學在此就不詳細描述，不過大多可以分為固定的一連串步驟，稱為 Graphics pipeline[13]。由於針對各個多邊形都有類似的計算步驟，故進行平行化的效益很大。

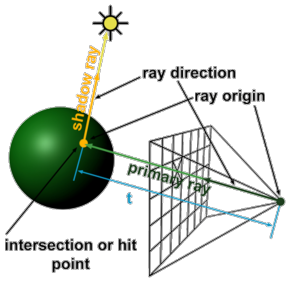
（二）兩種主要渲染方法

目前最常見的兩種渲染方法是「rasterization」和「ray-tracing」，示意圖如下[14]：

Rasterization



Ray-tracing



Rasterization 基本概念類似由頂點發出射線，找到和 projection plane 的交點，同時要處理的是可見性問題 (Visibility problem)。由於其計算過程較容易，但並不符合光線的物理特性，故主要用在需要 real-time 的應用上，例如遊戲、VR 等。

Ray-tracing 則是模擬真實世界的光線運作方式來進行渲染，從鏡頭射出光線，在物體表面執行數學模型描述的光學現象模擬，以達到更為逼真的效果。由於運算量相對很大，主要用在 pre-rendering 的應用上，例如電影特效、動畫等。

（三）「攝影機」、座標轉換、投影

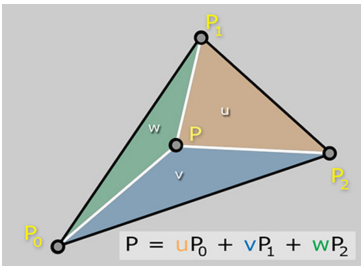
事實上「攝影機」的最簡單實作就是一個 4x4 的矩陣，這是由於攝影機的概念主要是為了對頂點做座標轉換，故我們會在攝影機中存一個 4x4 矩陣，稱為「World to Camera」，也就是將頂點在 world coordinate 轉換到 camera coordinate。如果要反過來轉換的話可以對該矩陣做反矩陣運算，即可從 camera 轉換回 world。座標轉換到 camera coordinate 後，便可以大幅降低投影的計算量。

---

[13] https://en.m.wikipedia.org/wiki/Graphics_pipeline
[14] https://www.scratchapixel.com/

（四） 重心座標 (Barycentric coordinate) 和內插

對於三角形而言，內部任意一點可以表示成下圖關係。



其中 u, v, w 分別為三個三角形佔整個三角形的面積比例，(u, v, w) 稱為重心座標。若每個頂點都帶有特定的量，例如貼圖座標、法向量等，那我們可以對三角形內每個點內插出其對應值。

值得注意的是內插還可分為直接內插和「透視正確」內插，在效能和視覺效果之間作取捨，關於透視正確內插的數學推導可參考相關書籍[15]。

（五） 貼圖和內插的透視正確性

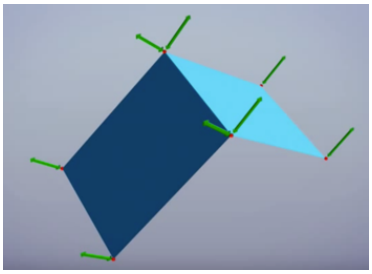
對於有提供貼圖座標的模型，每個多邊形的頂點會有一個二維的貼圖座標，介於 (0,0) 和 (1,1) 之間，代表的是該頂點對應到貼圖上的相對座標。對於三角形內的點可以選擇使用直接內插或透視正確的內插，視覺效果差異如下：

（六） 頂點法向量、平滑渲染

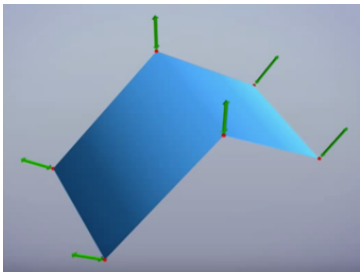
直覺上法向量應該是針對「面」的特性，但是在渲染的領域更喜歡在頂點上描述法向量，其中一個原因是由於這個做法讓我們得以控制面與面之間的平滑度。

由於每個三角形都是一個平面，頂點似乎應該有相同的法向量。但實際上比不一定需要相同，若我們根據三個頂點的光影來內插三角形內部的話，就可以達到平滑的效果，這個做法稱為 Gouraud Shading[16]。示意圖如下：

Flat Shading



Gouraud Shading

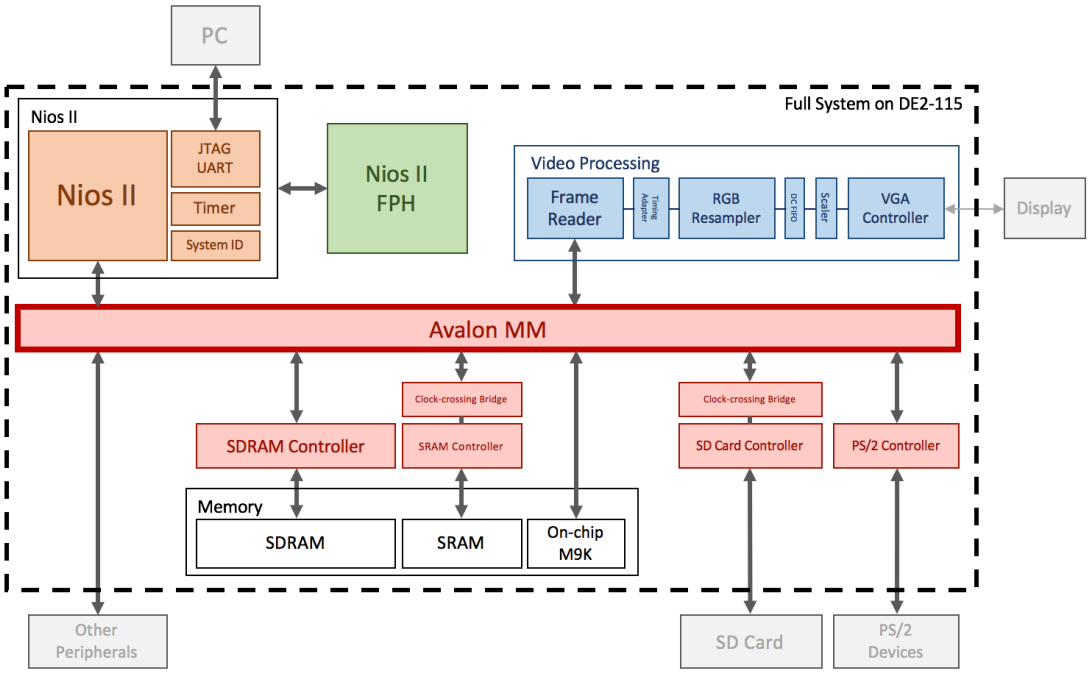


[15] Tomas Akenine-Moller. (2008). *Real-Time Rendering, Third Edition*. A K Peters/CRC Press.

[16] https://en.m.wikipedia.org/wiki/Gouraud_shading

五、硬體架構

（一） Block Diagram

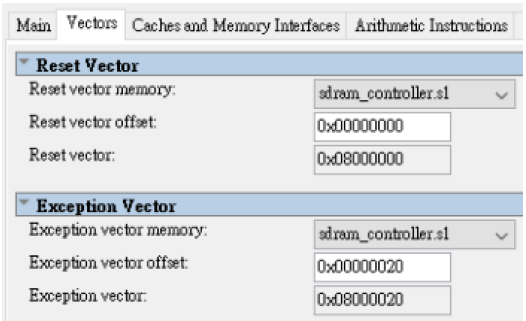


（二） 記憶體部分

記憶體主要使用兩種，分別針對不同需求使用：

1. SDRAM

從 DE2-115 的規格來看，SDRAM 相對 SRAM 的優勢是容量大（128MB ↔ 2MB），而由於記憶體中除了需要存程式本身以外，程式執行部分也會需要較大的記憶體（讀取模型、貼圖時使用），故我們在 Qsys 內將 Nios II 的 Reset vector 和 Exception Vector 設定在 SDRAM 上，如下圖：



如此一來在 Eclipse 上傳程式後，Nios II 會直接從 SDRAM 執行。

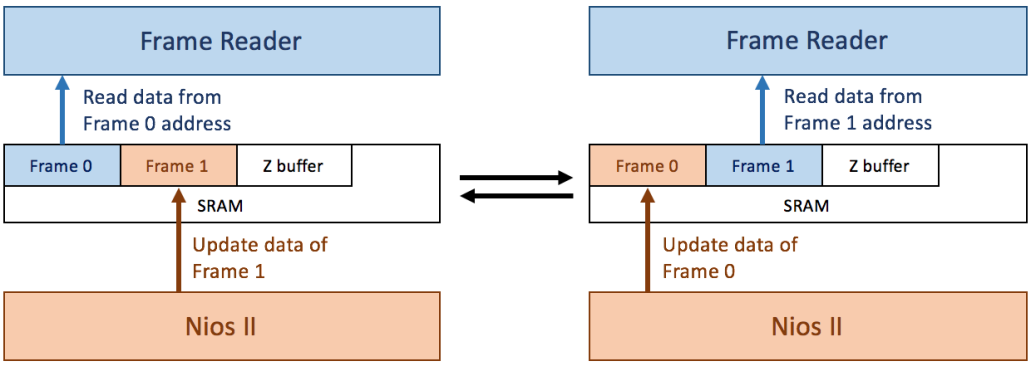
2. SRAM

SRAM 相對於 SDRAM 的優點則是，不需要像 SDRAM 需要經常進行更新產生不可避免的延遲，故在同樣的 clock 下 DE2-115 的 SRAM 會比 SDRAM 來得快，所以我們將他應用在渲染的 buffer 上，如 frame buffer 和 z buffer 等。但是由於 SRAM 的容量較小，故我們必須降低螢幕解析度。

稍微計算一下我們的解析度上限：

首先，frame buffer 每個像素需要 32bit（RGBA，但 A 不使用）。

而我們使用 2 個 frame buffer 來避免更新時螢幕閃爍，在顯示一個 buffer 時，對另一個 buffer 進行更新，如下圖：



最後 z buffer 的每個像素存有 IEEE754 single-precision 浮點數，故為 32bit。故螢幕解析度的像素數不能超過 2MB/12B，解析度上限約為 400x300。

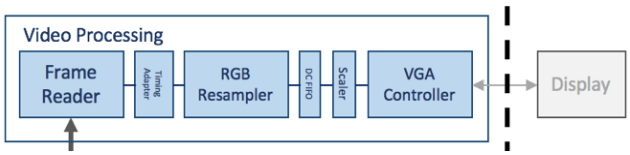
最後我們採用的解析度是 160x120，其中主要是效能需求，雖然非常低但可以達到 real-time 的效果，讓我們得以平順地用滑鼠、鍵盤操控視角。

不過值得注意的是，根據 datasheet[17][18]的 spec 以及經過測試後，發現 SDRAM 可以支援到 DE2-115 的 PLL 輸出的上限 120MHz，但 SRAM 只能到 60MHz。故 SDRAM 也許不一定會比 SRAM 慢，不過基於 SDRAM 已經用來執行程式的理由，許多傳輸頻寬會在程式使用上，故我們仍選擇在 SRAM 上面進行 frame buffer 和 z buffer 的操作。

[17] http://www.issi.com/WW/pdf/61WV102416ALL.pdf
[18] http://www.issi.com/WW/pdf/42S16320B-86400B.pdf

（三）影像輸出

硬體中影像部分在 Block diagram 的右上角：



這部分主要來自 Altera 提供的 VIP，彼此之間使用 Avalon-ST 來進行影像傳輸，簡單描述一下各部分的功能：

1. **Frame Reader**

   從 SRAM 的 frame buffer 中讀取 frame 的資料，轉換成 Avalon-ST 格式。我們使用的解析度是 160x120。

2. **Timing Adaptor**

   修正 Frame reader 和 RGB resampler 之間的一些和 Avalon-ST 相關的 timing 問題。

3. **RGB Resampler**

   從 Frame reader 得到的是 24bit 的色彩資料，後面的 VGA Controller 則使用 30bit 的色彩資料，可以使用此 resampler 來進行轉換。

4. **Dual-clock FIFO**

   VGA Controller 在 640x480 的情況下必須使用 25MHz，Frame reader 則使用 120MHz 故需要一個 FIFO 來協調不同 clock domain 之間的傳輸。

5. **Scaler**

   Frame reader 傳送過來的 Avalon-ST 是 160x120，需要藉由 scaler 放大 4 倍成 VGA 的解析度 640x480，是 VGA 標準中的最低值。

6. **VGA Controller**

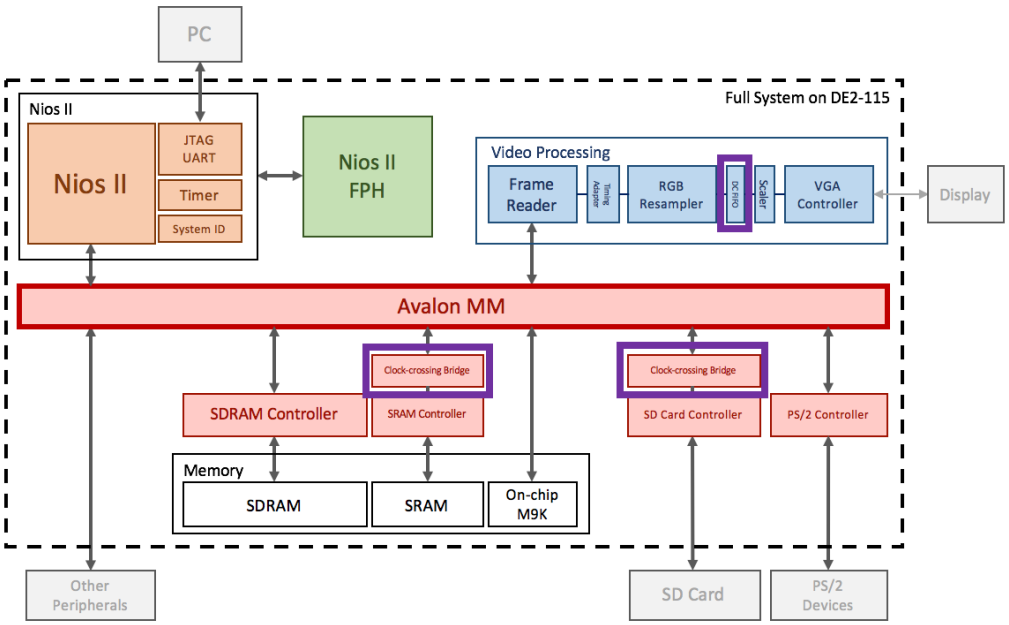
   顧名思義就是控制 VGA，我們使用的解析度是 640x480。

（四）不同 clock domain 間的協調

前面提到，Nios II 和 SDRAM 都能在 120MHz 下使用，而 SRAM 則只能在 50MHz 下使用，故在連接到 Avalon MM bus 上時，需要先進行不同 clock domain 間的協調。

這部分 Altera 也很方便的提供了 Clock-crossing Bridge[19] 能夠連接不同 clock domain 的 Avalon MM 介面，讓我們得以根據不同模組使用不同的 clock。至於連接不同的 Avalon-ST 介面則是使用 Dual-clock FIFO。

這次實驗中有三個地方（紫色部分）使用到這些模組：



分別為：

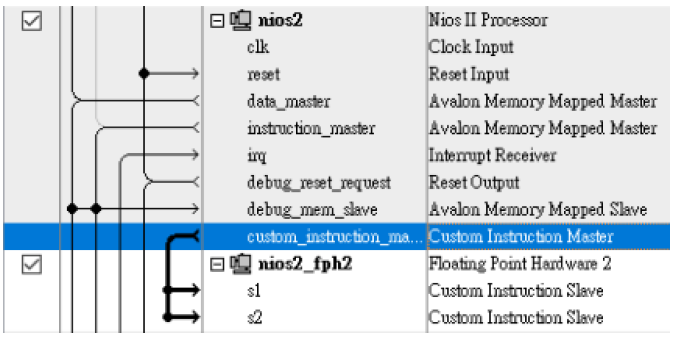
1. SRAM @50MHz ↔ Avalon-MM @120MHz
2. SD Card Controller @50MHz ↔ Avalon-MM @120MHz
3. Frame Reader @120MHz ↔ VGA Controller @25MHz

[19] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_embedded_ip.pdf

（五） Nios II 和簡易的硬體加速

前面提到 Altera 提供 FPH2，加速浮點運算，對本專題來說非常有用，而加速的效果也很明顯，詳細比較參考七、效能比較。

至於要如何在 Nios II 上加入 FPH2 呢？其實非常簡單，只要在 Qsys 中將 Nios II 的 Custom Instruction Master 連接到 FPH2 的 Custom Instruction Slave 即可，如下圖：



六、軟體架構

（一） 基本概念

渲染在軟體上看來，基本上會有場景，場景中有物件和光線，還有一台攝影機，而渲染結果即為攝影機所「看見」的畫面，所為「看見」其實就是前面提到的座標轉換、投影等過程。所以在軟體實作上也會有數個類別，分別對應到這些概念。

（二） 程式架構

由於我們使用的是 C，故沒有 class 和 member function 等功能，只能使用 struct，不過基本架構還是差不多的。首先根據上一節的描述，我們需要定義幾個 struct，由於 C 沒有 namespace 的功能，故我們在每個名字前面加上相同的前綴「ren」代表 render：

1. renScene：場景

每個場景可能有多個物件。

2. renObject：物件

每個物件中存有以下資料：

(1) 多邊形：每個面為三角形，由三個頂點座標表示。

(2) 頂點座標

(3) 頂點法向量：可用來計算 smooth shading。

(4) 頂點貼圖座標：每個頂點對應到的貼圖座標，範圍 (0, 0) 到 (1, 1)。

(5) 貼圖：一個 unsigned char 陣列，以 RGB 格式儲存整張貼圖。

3. **renCamera：攝影機**

一個攝影機事實上就是一個 4x4 矩陣，如前文所述。

此外我們也實作其他輔助的函數來移動攝影機，這其中就是許多的矩陣運算。注意到我們使用的旋轉方法是基於 Euler rotation[20]而不是 Quaternion[21]，所以可能會有 Gimbal lock[22]的現象產生，解決辦法也很簡單，我們的操作上並不包含三個旋轉軸向的「roll」方向，這也是一般遊戲操作中較少使用的旋轉軸。最後我們實作了 WASD 平移操作，以及 IJKL 旋轉視角和繞著 Y 軸旋轉。

4. **renRenderer：渲染器**

負責根據給定的場景、物件、渲染器來渲染至螢幕上。基本上就是執行一個簡單的 Graphics pipeline。此外還有初始化 SRAM 中的 Frame buffer、Z buffer 等功能。

除了這些渲染相關類別以外，還有其他輔助的部分：

1. **renLoop.h**

渲染是一個一直重複的動作，獨立出來是為了增加可讀性和方便維護。

2. **renInput.h**

負責監聽滑鼠、鍵盤事件，這部分使用到 Nios II 的 IRQ 功能，會在下節描述。

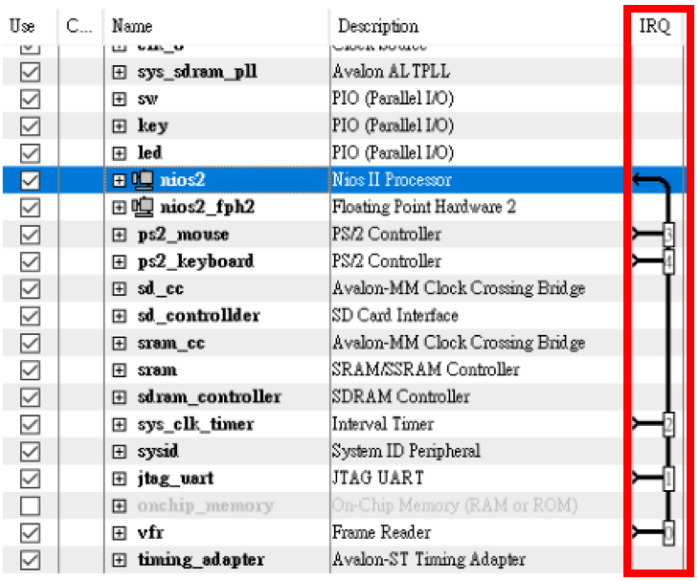
3. **objLoader.h、texLoader.h**

分別負責讀取模型檔案（.obj）和貼圖檔案（.png）。

[20] https://en.m.wikipedia.org/wiki/Euler_angles
[21] https://en.m.wikipedia.org/wiki/Quaternion
[22] https://en.m.wikipedia.org/wiki/Gimbal_lock

（三）使用 Nios II IRQ[23]

首先要在 Qsys 中連接 IRQ sender 和 IRQ receiver，在這次專題中分別為 PS/2 Controller 和 Nios II，如下圖：



接著要到 Nios II 專案 BSP 中的 system.h 中，找出 PS/2 Controller 對應的 IRQ 編號：

```
#define ALT_MODULE_CLASS_ps2_mouse altera_up_avalon_ps2
#define PS2_MOUSE_BASE 0x120010d8
#define PS2_MOUSE_IRQ 3
#define PS2_MOUSE_IRQ_INTERRUPT_CONTROLLER_ID 0
#define PS2_MOUSE_NAME "/dev/ps2_mouse"
#define PS2_MOUSE_SPAN 8
#define PS2_MOUSE_TYPE "altera_up_avalon_ps2"
```

可以得知滑鼠的 PS/2 Controller 編號為「PS2_MOUSE_IRQ」。

接著再使用 sys/alt_irq.h 中的函數 alt_irq_register()，註冊 IRQ 所需要觸發的函數：

```
alt_irq_register( PS2_MOUSE_IRQ  edge_capture_ptr, handle_mouse_irq );
```

監聽來自滑鼠的事件　　滑鼠事件觸發時執行的函數

注意到 handle_mouse_irq 這個函數必須定義如以下格式：

```
void (*alt_isr_func) (void* isr_context)
```

context 的部分在這次專題中並沒有使用到，但是針對更為複雜的情況則可以傳入目前程式的許多狀態，十分方便。鍵盤方面也是和滑鼠完全一樣的做法。

至於 Nios II 支援的 IRQ 上限可以達到 32 個[24]，本實驗只用到 5 個支援 IRQ 的模組。

---

[23] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2.pdf
[24] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf

（四） 從 SD 卡讀取檔案

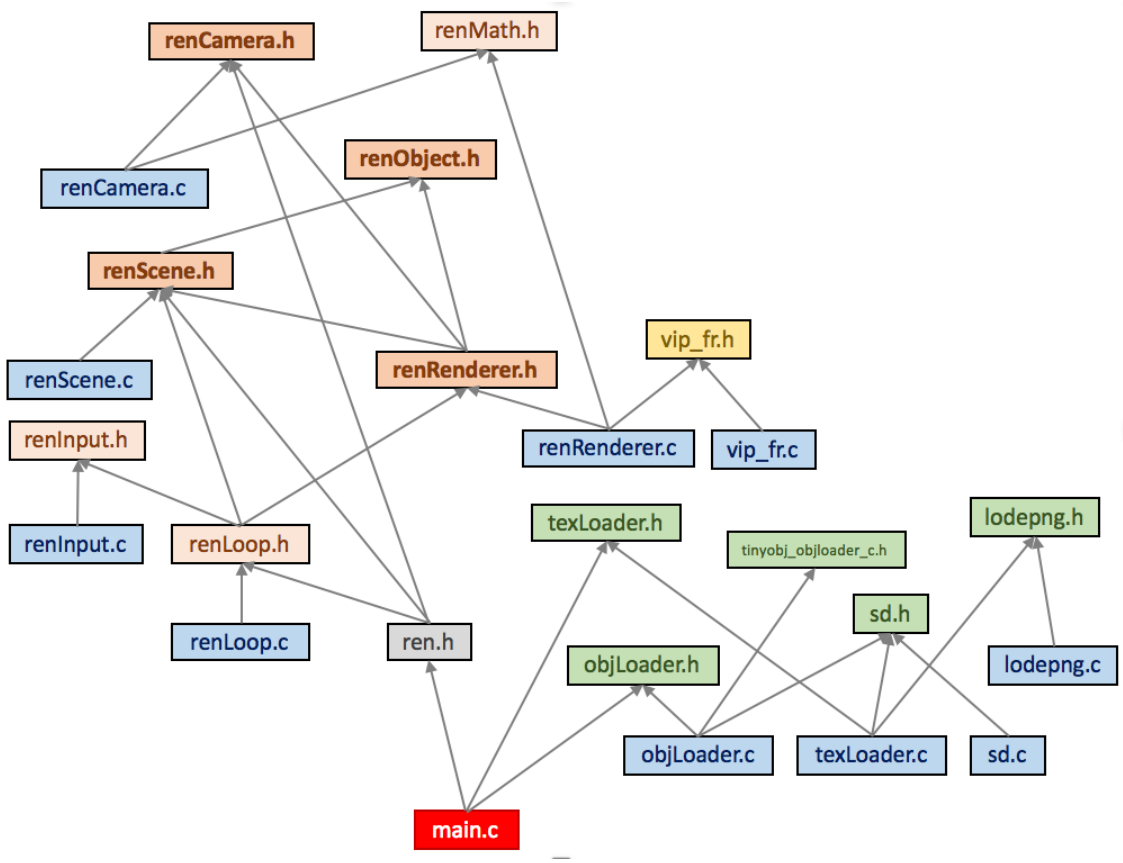
Altera University Program 中的 SD Card Controller，提供了 C 的 API，這部分可以直接參考官方文件[25]。

（五） Parsing .obj and .png

要 parse 這兩種格式，其實可以自己撰寫 parser，不過為了開發效率，就從 Github 上面找開源專案了。我們針對這兩種格式都使用 C89 撰寫的 parser，分別為 tinyobjloader-c[26] 和 lodepng[27]。

（六） 相依性圖表

下面的圖描述了原始碼和標頭檔間的 include 關係，供參考：



[25] ftp://ftp.altera.com/up/pub/Intel_Material/16.1/University_Program_IP_Cores/Memory/SD_Card_Interface_for_SoPC_Builder.pdf
[26] https://github.com/syoyo/tinyobjloader-c
[27] https://github.com/lvandeve/lodepng

七、效能比較

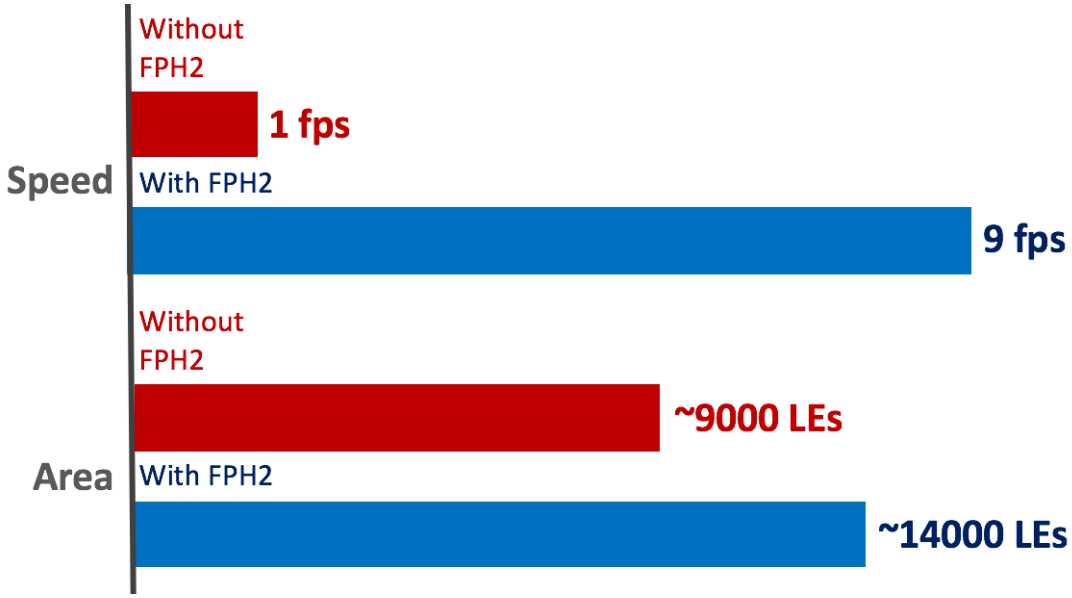
（一）使用 Floating-point Hardware 2 的差異

前面提到 FPH2 可以提升浮點運算的效能，對加速渲染這種使用大量浮點運算的應用有很大的幫助，這裡比較使用 FPH2 後的效能以及面積差異：

使用的場景是鏡頭繞著跑車模型 camero.obj 旋轉。

模型有 355 個頂點，690 個面。測試時使用貼圖和平滑渲染。

測試結果如下：



注意到 framerate 提升了將近 10 倍，但是 area 只增加了約 50%，故可說明在這次實驗中，加上 FPH2 所增加的面積是一個不錯的 trade off。

（二）提升 Nios II 和 SDRAM 的 clock speed 的差異

一如前文所述，Nios II 在 Cyclone IV 上可以支援到 160MHz，但由於 PLL 的限制我們只能使用到 120MHz。在此比較由 50MHz 提升到 120MHz 的效能差異：

使用的場景和上一節相同：

注意到當 clock speed 上升到一定程度時，提升的 framerate 比例就不太增加。推測這是由於此時的效能瓶頸從 SDRAM 轉移到 SRAM 上，因為 SRAM 的 clock speed 只能達到 50MHz，故效能提升幅度降低。

八、展望

這次實驗原先還預期做出類似 GPU 架構的硬體，不過因為時間關係還來不及做出來。未來希望能真的在 FPGA 上實作簡單的 GPU 架構，真正了解 FPGA 所帶來的硬體彈性優勢，也能提升自己對平行運算的體會和熟悉度。