Chapter 4 Just-In-Time Rendering Scheme of Object Movies

在第三章我們考量了環物影片與 H.264 的特性，提出了一套環物影片排列的方法以便於 H.264 壓縮，接下來爲了使壓縮後的環物影片能夠順暢的播放，本章將介紹多層暫存的架構(multi-layer cache)來提高使用者觀賞環物影片時的互動性。並根據環物影片的特徵，進一步改善的環物影片壓縮與解壓縮的效能。最後，結合 single-instruction multiple data (SIMD)技術與解碼器的最佳化達到 Just-In Time(JIT) Rendering 目標。

4.1 Multi-Layer caching structure

前面一節提出了排列環物影片的方式，使得二維的環物影片轉爲一維之影像序列，同時又能提升壓縮後的影像品質與考量環物影片播放時的順暢度。此外，爲了達到環物影片即時描繪(just-in-time rendering, JIT)，本節將提出一個 cache 的架構，透過此一架構能夠進一步有效降低解碼器的負擔，讓使用者可以順暢地觀賞環物影片。

環物影片與一般視訊影片觀賞上最大的不同在於環物影片是依據觀賞者的轉動決定播放的影像，沒有一定的順序屬於隨機播放；而視訊影片則爲循序性的播放，雖然可以倒轉或快轉但通常只能移動到特定的 I-frame。此外，視訊影片播放過後就可以丟棄，不需要存放在記憶體中，然而環物影片播放過的影像很有可能馬上又會需要播放。如果每次都需要從 bitstream 中重新解壓縮或者把所有

解過的影像都存放在記憶體裡的話，對於使用者的電腦而言都會是個沈重的負擔。爲了達到環物影片 JIT rendering 的播放，對於環物影片播放器快速隨機存取(random access)的能力要比一般視訊影片播放器高上許多。一般視訊影片播放器通常只提供特定幾張影像的隨機存取，但是使用者觀賞環物影片時有可能會觀賞到任意一張的影像，因此播放器必須能夠正確的播放使用者想要觀看的影像，解碼器也必須配合從 bitstream 裡解出該張影像。

爲了達到這樣的目的，我們建立一個多層 cache 架構解決隨機存取的問題。在介紹 cache 架構之前，先簡單的回顧 H.264 解壓縮的流程(如圖 4.1)，H. 264 的 bitstream 是由許多 NAL unit 所構成，當解某一張影像時必須先在 bitstream 裡找到該影像所在的 NAL unit 位址，再經由 CABAC 或 CAVLC entropy decoding 解出壓縮參數和 MB 係數，然後透過 inverse quantization 和 inverse transform 得到 residual coefficient，最後跟參考的影像做 motion compensation 之後就可以還原出完整影像。此外，H.264 會在影像輸出之前利用 in-loop deblocking 消除 block distortion 現象。

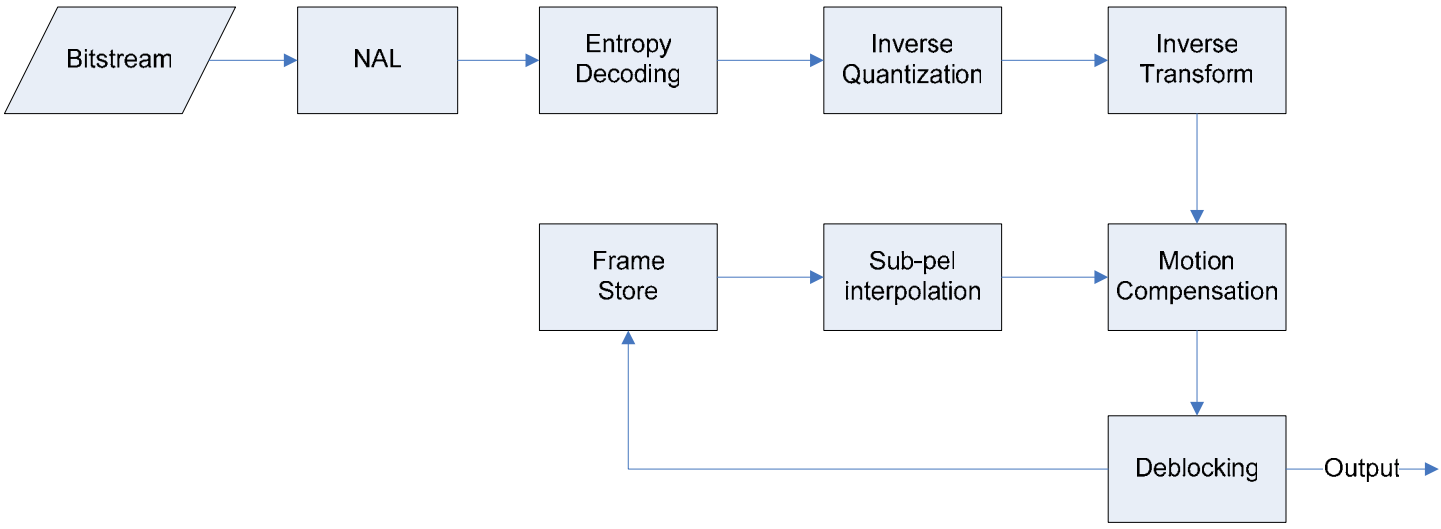

圖 4.1 H.264 解壓縮流程圖

根據[23]的分析(如圖 4.2)，上述的解碼流程最花時間的部分爲 motion compensation，其次爲 integer transform 和 dequantization，因此會以節省這些步驟最爲設計 cache 架構的依歸。

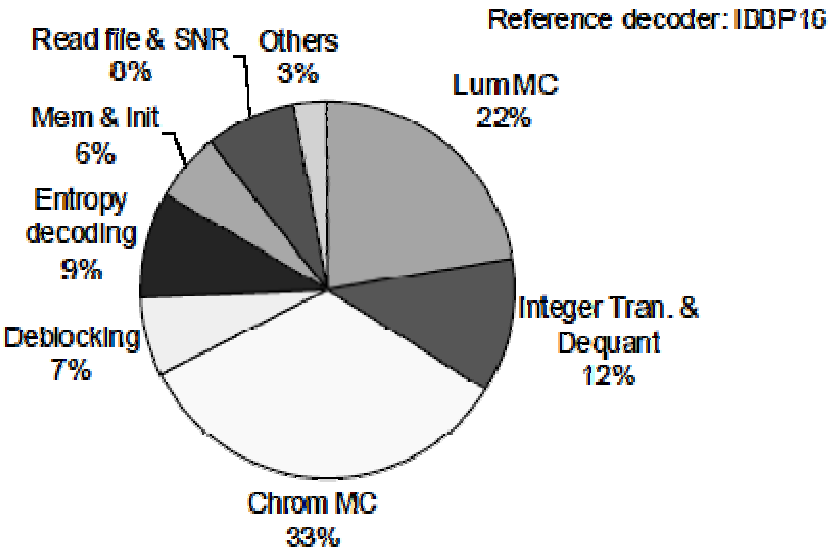

圖 4.2 H.264 reference software 時間分析圖

在[27]詳細表列了 H.264 reference software decoder 執行時所需記憶體的大小，見表 4.1，重建後的影像以及用來當作參考的影像需要最多的記憶體，其次則爲 motion vectors 和一些暫存的資料，cache 的容量時必須視使用者的環境最適當的配置。

Buffer name	Formula	QCIF w=176, h=144, n=1	CIF W=352, h=288, n=1
Reconstruction frame	1.5*w*h	38016	152064
Reference frames	1.5*w*h*n	38016	152064
Slice map	w/16*h/16*4	396	1584
Reference indices	w/16*h/16*4	396	1584
Motion vectors	w/16*h/16*64	6336	25344
Intra-prediction modes	w/16*h/16*16	1584	6336
CBP values	w/16*h/16*4	396	1584
MB types	w/16*h/16	99	396
QP values	w/16*h/16	99	396
CAVLC coefficient counts	w/16*h/16*16*1.5	2376	9504
MB temp data	~2048	2048	2048
Constants	~2048	2048	2048
Total	(n+1)*1.5*w*h+ w/16*h/16*118+ 4096	91816	354952

表 4.1 H.264/AVC baseline profile decoder 記憶體需求(單位: byte)

基於以上分析，我們建立了一個多層的暫存架構(multi-layer caching structure)來克服環物影片隨機存取的問題，並減輕解碼器的負擔達到 JIT rendering 播放的目的。因爲環物影片的觀賞跟一般的影片不同，使用者很可能會隨時觀看之前看過的影像，因此把已經重建後的影像暫存起來，當使用者需要觀看時 rendering engine 會先到 cache 中搜尋是否有重建後的影像存在，若沒有則再要求解碼器重

新解壓縮該張影像。如此將會大幅減少解壓縮的時間，提供更順暢的觀賞效果。因此在架構之中，最上層的 cache 直接存放重建後的影像(reconstructed frames)，以我們的環物影片來說暫存一張影像需要 786432 bytes (512x512x3)，因此存放重建後的影像需要大量的記憶體，所以 cache 的大小需要視不同的使用者環境做調整。

此外，爲了縮短 motion compensation 和 inverse transform、inverse quantization 的時間，最簡單的作法是將這些結果都先暫存起來，但暫存這些結果需要的記憶體空間相當於儲存一張完整的影像。因此，中間層的 cache 僅會儲存曾經解出過的 motion vector 和 reference index 以節省一些解壓縮的時間。但只有 motion vector 和 reference index 是不足以解壓縮一張影像，要重建某張影像還是必須先到 bitstream 中找到影像所在 NAL unit，而爲了能夠快速在 bitstream 中找到所需的 NAL unit，最底層的 cache 會儲存每張影像所存放的 NAL unit，並透過 index table 記錄每個 NAL unit 的位址。以上 3 層的 cache structure 的完整示意圖如圖 4.3

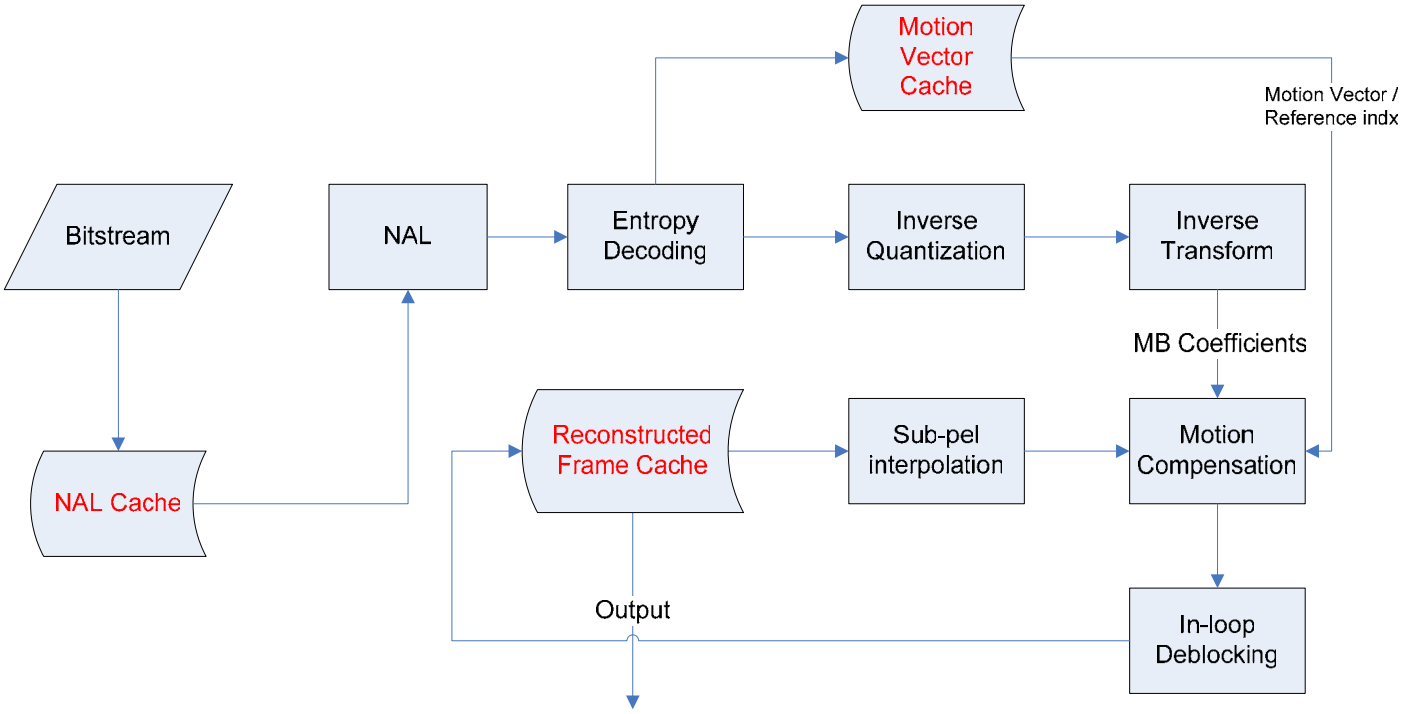

圖 4.3 Multi-layer Caching Structure

在此，最上層的 reconstructed frame cache 屬於 double-linked list，其管理機制是採用類似 FIFO 的方式，但依據環物影片二維的空間關係將目前觀看影像以及周邊的鄰近影像移至 cache 的最前面。若該張影像不存在於 cache 之中，則解壓縮之後則放入 cache 的最前面，並將他的鄰居依序的解出(若尚未解出)也放入 cache 之中。此外，reconstructed frame cache 的大小也會影響到播放的順暢度，越大的 cache 容量能夠提供越好的觀看效果，但需要的記憶體容量也是很可觀，必須針對最後觀看的平台做適當調整。中間的 motion cache 和底層的 NAL cache 因爲需要的記憶體較少，所以實做時會將所有資料存放在記憶體。

4.2 Foreground and Background Coding

在第二章提到增添式環場(Augmented Panorama)乃是將環場影像(Panorama)與環物影片(Object Movie)透過符合幾何一致性的方式自然的融合在一起[7]，給使用者有如真實一般的 3D 虛擬展示效果。將環物影片融合在環場影像之前，必須先對影像資料做去背前處理(segmentation)，使得物體的前景與背景分離，如圖 4.4，圖(b)爲物體 alpha 值的灰階影像，藉由 alpha 值可以將物體與背景作 alpha blending。雖然目前有幾種影像格式支援存放影像的 alpha 值，如 png…。但是 H.264 只支援的 YC_bC_r video color space，Y 代表影像的亮度(luma)，C_bC_r 代表影像的藍、紅的色度(chroma)。雖然在 H.264 的延伸規格 Fidelity Range Extensions(FRExt)提出了 auxiliary pictures[25]，用以輔助影片產生 alpha blending 的效果，但考量目前編碼器跟解碼器架構無法同時處理環物影片和其 auxiliary pictures。爲了整體播放效率和處理便利性上，我們暫不考慮 auxiliary pictures，留待將來再進一步探討。

(a) (b)

圖 4.4 (a)環物影片的影像,(b)前景/背景 mask 影像

爲了便利壓縮含有 alpha mask 之環物影像，我們先對環物影片和 alpha mask 影像的做前處理。因爲 H.264 在新的 FRExt 規格中支援了更高精度的 color space，例如 Y C_bC_r 422 和 Y C_bC_r 444，利用這個特性首先將環物影片和 alpha mask 影像從 RGB 轉換成 YC_bC_r 422。 事實上，alpha mask 影像只是一張灰階影像，轉換成 YC_bC_r 之後只需要保留 Y channel 的資訊就足以用來做 alpha blending。因此，將環物影像的 C_bC_r 兩個 channel 合併單一個 channel，加上 mask 影像的 Y channel 合併成一個新的 YC_bC_r444 的影像(如圖 4.5)。如此一來，將環物影片與 alpha mask 影像合併成一個 YC_bC_r444 格式的檔案後就可直接經由 H.264 的編碼器壓縮，同時也減少了影像資料量的大小。

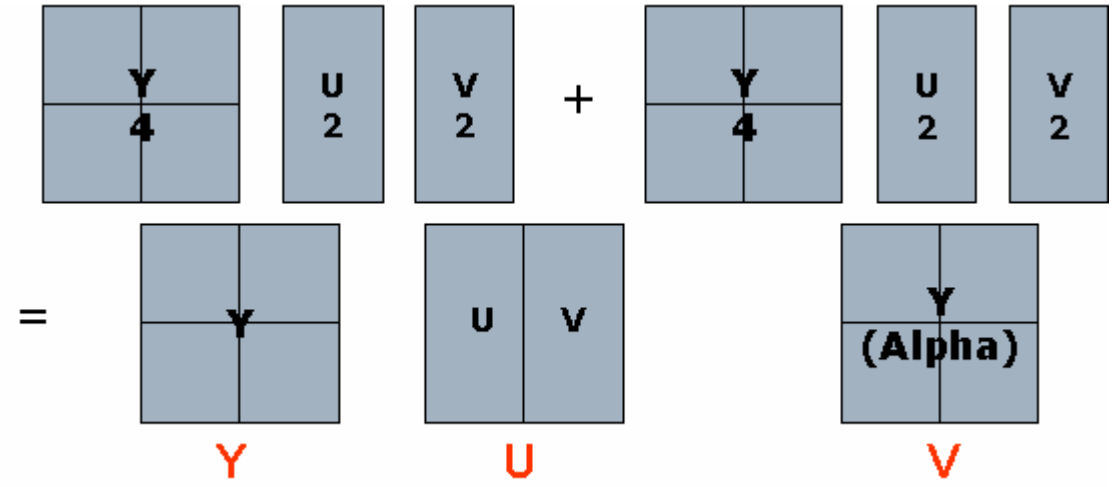

圖 4.5 環物影片與mask 影像合併示意圖

以這種方式編排過的環物影片，以一般的 H.264 解碼器還是可以還原影像，但影像的顏色會混亂且有鬼影，對於博物館在虛擬展示時可以提供最基本的保密功能。然而，這樣合併的缺點在於 U、V 兩個 channel 的影像不同，壓縮之後的 PSNR 會略微降低，實際測試降低的幅度約 0.05dB，對於實際觀看的品質並不會有太大的影響。

4.3 The Performance Improvement of Compressing Object Movie

3.3 節解決了環物影片跟 alpha mask 影像如何同時壓縮的問題。本小節將進一步利用環物影片的 alpha 資訊來加速編碼與解碼的過程，尤其 H.264 壓縮演算法的複雜度比之前的視訊壓縮標準都要複雜許多，如果可以降低壓縮與解壓縮的時間，將有助於環物影片的製作與播放。

新的視訊壓縮標準(如 MPEG-4 或 H.264)在壓縮影片時若能夠獲知其前景與背景的資訊，則通常可以利用該資訊針對前景與背景作不同的處理以獲得更高的壓縮效能，像 MPEG-4 的 shape coding[21]或是 H.264 的 Arbitrary slice ordering (ASO)在前景或背景的 slice 給予不同的 QP[18]。在[24]，Lin et. al.提出一個在知道前景與背景資訊下增加影像視覺品質的方式，藉由設定 RD optimization 不同的 λ 值達到控制前景與背景壓縮品質的目的。

觀察環物影片可以很明顯的發現，物體前景的部分大概只佔了整張影像二分之一左右，以 H.264 這種 block-based 的壓縮標準來處理環物影片時，表示有將近一半的時間是用在編碼或解碼背景的部分，而這些背景 MB 處裡的結果對於虛擬展示的結果來說是一點都不重要，因爲背景會在 alpha blending 之後被略去。所以背景的 block 最好的處理方是就是完全忽略不壓縮，不但可以減少壓縮的時間，也可以增加解壓縮的速度。在 H.264 規格中有定義了 skipped macroblock 型態，將沒有任何資料需要壓縮的 MB 標示成 skipped 不予處理，但在 reference software 僅有 B-frame 能夠包含 skipped macroblock。我們改良 reference software 不管是哪種 frame type 都能夠將屬於背景的 MB 直接標示成 skipped。

在 reference software 實作的 H.264 架構裡，每個 MB 的資料在寫入 bitstream

之前，會加入一些檔頭的資訊，包括 type、QP、CBP、skipped…等等。壓縮時我們會先計算每個 MB alpha 值的總和，看是否超過一個 threshold 來判斷該 MB 屬於前景或背景。然後將這個資訊寫入 MB 的檔頭，將來在解壓縮時就能夠藉由這個資訊得知 MB 是屬於前景還是背景。在實做時，屬於背景的 MB 因爲不影響 alpha blending 的結果，因此不管在壓縮或解壓縮都可以略過不需要做任何處理。

4.4 Optimization Decoder

H.264 為了達到在低 bit-rates 還能夠維持良好的視訊品質，採用了許多複雜的演算法，譬如 H.264/AVC baseline decoder 的複雜度比 H.263 baseline decoder 要高 2-2.5 倍，因而導致不管在編碼器或解碼器對於電腦的運算能力都有很高的要求。以目前 reference software 的執行效率來看，無法滿足虛擬博物館對於 just-in-time rendering 的要求，以 reference software 9.0 為例平均解壓縮一張環物影像需要 500ms 左右。因此本章最後將探討如何最佳化解碼器以達到即時互動之虛擬展示。

現代的微處理器為了提升多媒體應用程式效能，大多支援多媒體指令集 (multimedia instruction)。例如 Intel 提出 single-instruction multiple data (SIMD) execution model，MMX, SSE and SSE2 技術可以在單一指令同時執行多個計算達到提升效能的目的。SIMD 技術利用 longer registers 能夠同時執行多個算數或邏輯運算，如 MMX 的 64-bits registers 或者 SSE 的 128-bits registers。在[23][27][28]為了最佳化 H.264 decoder 均採用這樣的技術達到 real-time decoding。

在本文系統藉由重新編寫了 H.264 JVT reference model software 9.0，包括最佳化程式模組和 memory alignment，同時結合 Intel® Integrated Performance Primitives library(Intel® IPP)，透過 Intel® IPP 中的 library，我們可以利用 Intel MMX 或 SSE 技術改善 inverse transform、inverse quantization 和 deblocking filter 執行效率，獲得 decoding 效能大幅的改善。