中華民國第 63 屆中小學科學展覽會

作品說明書

高中組　電腦與資訊學科

052514

魔鬼藏在細節中－實時影像隱藏浮水印

學校名稱：桃園市立武陵高級中等學校

作者：	指導老師：
高二 謝昀宸	周威宏

關鍵詞：動態影像、隱藏浮水印、著作權保護

摘要

本研究旨在透過 DCT-SVD 演算法，即時對動態影像，嵌入人眼幾乎不可見的浮水印。經過測試，該演算法具有一定的抗攻擊性，可以保障影像受到他人壓縮、裁剪等操作後，能還原嵌入的浮水印資料。

嵌入的浮水印有多種用途。舉例來說，版權方可以在動態影像中，嵌入可辨識被授權者資訊的資料。當被授權者私自將影像給予未授權者，版權方可以透過還原浮水印的方式，追蹤違反授權的被授權者，並採取必要的法律行動，以保障版權方的權益。或者反過來思考，對於監視器畫面之類需要做為證物的影像，當偵測到浮水印遭到破壞時，就代表該影像經過人為修改，可能不具有法律效力。

壹、前言

一、 研究動機

電子遊戲是現今十分熱門的一種娛樂活動，軟體公司開發一款電子遊戲產品，需要耗費大量人力、時間及金錢。一款電子遊戲在正式發布前，通常會有一些內部測試版本，以驗證遊戲的玩法、遊玩體驗等是否合乎玩家需求。這些內部測試版本通常包含相關的開發工具，因此通常不會對外公開，以保護遊戲公司的智慧財產權，也能避免因未完成的內容導致遊戲公司聲譽、財務上的損失。

然而，不時可以看到新聞，關於電子遊戲的測試版本因各種意外洩漏，對遊戲公司以及開發人員產生巨大的損失 [1] [2] [3]。因此，我開始思考，遊戲公司如果遇到類似的洩漏事件，有沒有方法能簡易的追蹤洩漏的人是誰，如果能在事後向洩漏者做出懲罰，也許就能遏止洩漏事件的發生。

經過搜尋與整理資料發現，浮水印是一個較常使用的方式。其中，許多遊戲公司仍然使用肉眼可見的浮水印，在螢幕顯示使用者（開發人員）的資訊，用以追蹤。這種直接在畫面中嵌入影像的方式，會影響視覺觀感 [4]。隱藏浮水印人眼幾乎不可見，但在影像處理領域，大多針對靜態影像，並未對動態影像有實際的測試、應用。

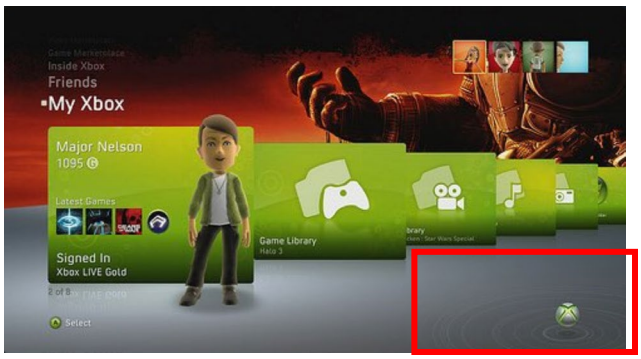


圖 1、XBOX 360 開發階段曾經用不顯眼的圓圈來追蹤使用者。

隨著科技進步，越來越多創作以動態影像的方式呈現，其中，前述的「遊戲開發」就是需要即時加上浮水印的使用案例。加上隱藏浮水印後，可以保障需要保密的影像被洩漏後，能夠追蹤洩漏者並維護智慧財產權，同時最小化對螢幕畫面的影響。因此，本研究探討是否有可能即時在動態影像中，嵌入隱藏浮水印。

二、 研究目的

（一） 探討隱藏浮水印的參數對於影像品質、抗攻擊能力、運算速度的影響。

（二） 實作即時影像浮水印的範例、測試執行速度及成效，並分析可行性及探討在實際生活中有哪些應用。

三、 文獻回顧

（一） 隱藏浮水印分類

1. 強健／脆弱：強健的浮水印能抵禦攻擊，影像被更改之後，能還原資料。
2. 可見／不可見：直接在影像上疊加文字或影像為可見浮水印，會影響視覺效果；不可見浮水印則透過不同技術、演算法，盡可能減低對視覺的影響。
3. 作用於空間域／頻率域：作用於空間域為直接修改像素資料，除了上述之可見浮水印之外，另有最低有效位（Least Significant bit, LSB）法，其特色為運算簡單、視覺上不易察覺，但無法抵抗如 JPEG 壓縮等攻擊。作用為頻率域通常會透過不同的轉換，如：離散傅立葉轉換、離散餘弦轉換、離散小波轉換等方式，轉換到頻率域，再添加資訊，最後進行逆轉換。其優點為較能抵抗不同種類的攻擊（強健性更高），缺點為實作較為困難、需要用較多的運算資源、運算消耗時間長。 [5]

（二） 動態影像隱藏浮水印

過去的研究主要是針對靜態影像的隱藏浮水印。動態影像實際上也是由一張張的靜態影像所構成，故靜態影像隱藏浮水印的演算法也可套用在動態影像上，即針對

動態影像中每一張影像，以靜態添加影像隱藏的演算法加上浮水印。然而這種做法是否能抵禦常見的裁剪、模糊以及影片壓縮攻擊，並未得到完整的測試、驗證 [6]。若要即時的對動態影像加上浮水印，運算速度也是需要納入考量的項目，若運算速度無法達成將輸入的動態影像即時添加浮水印，並輸出、呈現於螢幕，畫面就會是不流暢、無法使用及觀看的。

（三） 常見隱藏浮水印演算法

1. 離散餘弦變換（Discrete Cosine Transform, DCT）

離散餘弦轉換使空間域與頻率域能相互轉換。對一個 $m \times n$ 的影像進行 DCT，空間域 $f(x,y)$ 與頻率域 $F(i,j)$ 的對應關係為以下數學式：

$$F(i,j) = \phi(i)\phi(j)\sum_{x=0}^{n-1}\sum_{y=0}^{m-1} f(x,y)\cos\left[\frac{(2x+1)i\pi}{2n}\right]\cos\left[\frac{(2y+1)j\pi}{2m}\right]$$

$$f(x,y) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} \phi(i)\phi(j)\,F(i,j)\cos\left[\frac{(2x+1)i\pi}{2n}\right]\cos\left[\frac{(2y+1)j\pi}{2m}\right]$$

$$\phi(i) = \begin{cases} \sqrt{1/n}\ , i = 0 \\ \sqrt{2/n}\ , i = 1,2,\dots,n-1 \end{cases} \qquad \phi(j) = \begin{cases} \sqrt{1/m}\ , j = 0 \\ \sqrt{2/m}\ , j = 1,2,\dots,m-1 \end{cases}$$

頻率域 $F(i,j)$ 的索引 $0 \le i < n,\ 0 \le j < m$ 頻率由低到高分布。轉換到頻率域能將資訊嵌入到肉眼較難察覺的特徵當中，同時保持較好的抗攻擊性。因為人眼對中低頻的資訊較為敏感，常見的影像壓縮方式，如 JPEG，即是透過使用類似轉換，捨去高頻資訊來減少壓縮量。使用 DCT 轉換，將影像轉換至頻率域後，針對中低頻添加資訊，即可有效的抵抗相同類型的影像壓縮。 [7] [8]

2. 奇異值分解（Singular Value Decomposition, SVD）

奇異值分解是一種常用於矩陣的數值分析方法。奇異值分解可以將一個矩陣 $M_{m\times n}$ 分解為三個矩陣 $U_{m\times m}\,D_{m\times n}\,V^T_{n\times n}$ ，滿足以下數學式：

$$M_{m\times n} = U_{m\times m}\,D_{m\times n}\,V^T_{n\times n}$$

$$= \begin{bmatrix} u_{1,1} & \cdots & u_{1,m} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \cdots & u_{m,m} \end{bmatrix} \begin{bmatrix} \gamma_{1,1} & \cdots & 0_{1,n} \\ \vdots & \ddots & \vdots \\ \vdots & \cdots & \gamma_{p,p} \\ 0_{m,1} & \cdots & 0_{m,n} \end{bmatrix} \begin{bmatrix} v_{1,1} & \cdots & v_{1,n} \\ \vdots & \ddots & \vdots \\ v_{n,1} & \cdots & v_{n,n} \end{bmatrix}^T \quad \begin{cases} p = \min(m,n) \\ \gamma_{1,1} \ge \gamma_{1,1} \ge \cdots \ge \gamma_{p,p} \ge 0 \end{cases}$$

且滿足矩陣 $D_{m\times n}$ 非對角線上的元素皆為 0。

其中 $U_{m\times m}$ 為 $m \times m$ 階實正交矩陣； $V_{n\times n}^{T}$ 為 $V_{n\times n}$ 的轉置，為 $n \times n$ 階實正交矩陣，而 $D_{m\times n}$ 則稱為奇異值矩陣，為一個對角矩陣。影像處理上使用 SVD 的主要優勢為，SVD 可以分析影像中的主要構成部分，且數值大的奇異值在受到壓縮、干擾時有較高的穩定性。 [7] [9]

貳、研究設備與軟體

一、 系統、驅動程式及軟體

表 1、軟體規格表

名稱	版本	描述
Windows 11	25375.1	作業系統。
Python	3.9.13	人工智慧、資料分析等領域常用的高階程式語言。
Pytorch	1.13.1+cu117	用於深度學習的函式庫，更加快速且易用。
Torchvision	0.14.1+cu117	PyTorch 配套函式庫，提供易用的影像處理工具。
OpenCV	4.7.0	電腦視覺的函式庫，提供多種影像處理的工具。
Windows 11 SDK	10.0.22000	微軟公司提供給 Windows 應用程式開發者的軟體開發包。
Visual Studio	17.5.1	微軟公司提供的整合開發環境（IDE）。
MSVC	14.35.32215	微軟公司提供的 C/C++編譯器及函式庫。
Davinci Resolve	18.1.4	影片剪輯軟體。

備註：OpenCV C/C++ 由 4.7.0 原始碼自行編譯，開啟 /O2（偏好速度）優化。

二、 硬體設備

表 2、硬體規格表

名稱	型號
中央處理器 (CPU)	AMD Ryzen 7 5800HS
記憶體 (RAM)	DDR4 3200MT/s 48GB
圖形處理器 (GPU)	GeForce RTX 3060 Laptop 6GB

參、研究方法及過程

一、 研究流程

在閱讀了相關文獻之後，先用 Python 撰寫程式碼，測試靜態影像隱藏浮水印演算法是否能有效抵禦動態影像會受到的攻擊，並評估調整各項參數對結果的影響。找出最佳參數組合之後，用 C++ 撰寫範例程式，測試靜態影像演算法用於動態影像上的執行速度及成效。

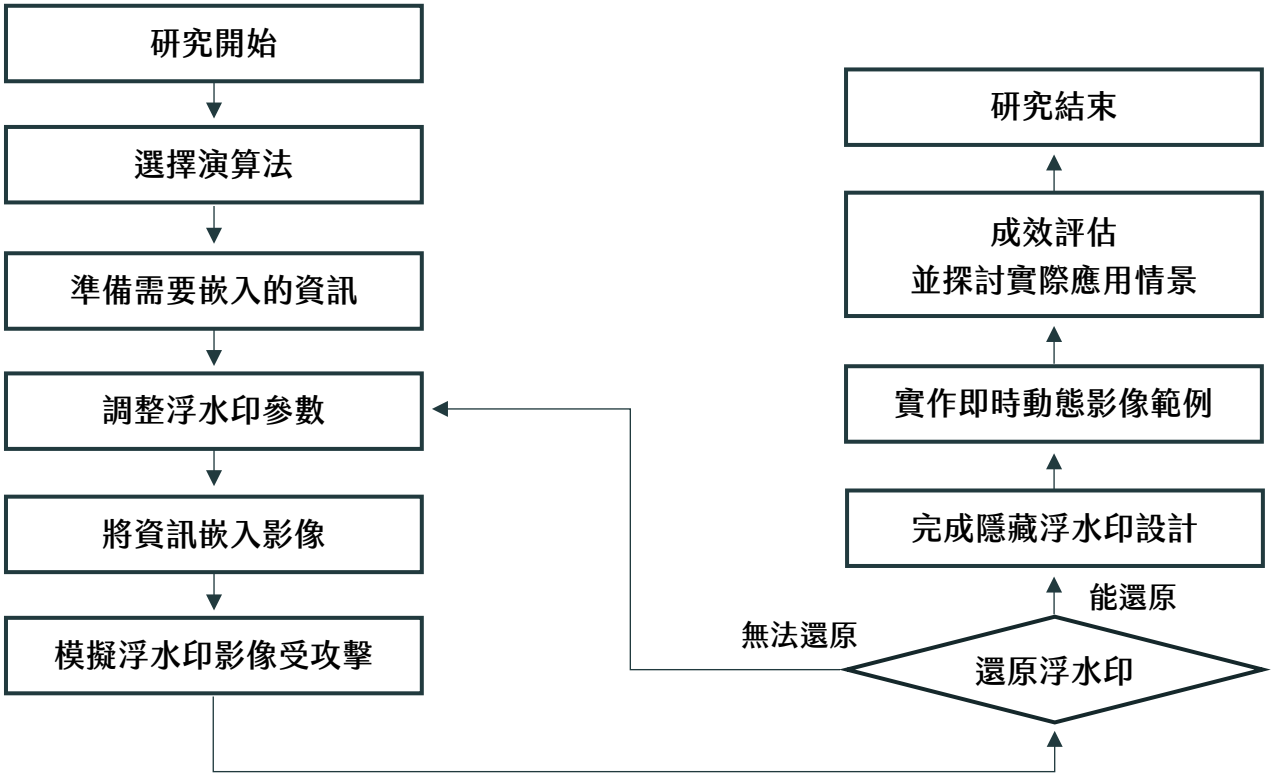


圖 2、研究流程圖

二、 研究過程

(一) 選擇演算法

本研究在閱讀了不同演算法組合的論文 [7] [9] [10]，選擇 DCT 作為空間域及頻率域的轉換，並以 SVD 提取關鍵資訊。因影像的主要由中低頻構成，故經由 DCT 轉換得到影像的頻率組成，再使用 SVD 分解時，就算影像的高頻部分被修改，奇異值矩陣的主要數值也不會受到太大的影響（此時奇異值矩陣的主要數值是表現影像的低頻部分）。兩種演算法組合的浮水印為肉眼幾乎不可見、具有一定的抗攻擊性，較符合隱藏浮水印的需求。 [11] [7]

（二）準備需要嵌入的資訊

如果欲嵌入的是 0 和 1 所組成的黑白圖片浮水印，則不用轉換；若欲嵌入的資訊是數字或字串，皆需要轉換成二進位序列形式，才能嵌入到目標圖片中。數字可以直接轉換成數個位元組儲存，如以 4 位元組（32 位元）儲存的帶正負號整數可以表示 -2,147,483,648 到 2,147,483,647 的數字；字串可以用 ASCII、Unicode 等編碼，將文字轉換成二進位形式，如 ASCII 編碼中，每一個字元以 1 位元組儲存，A 對應的數字是 65，二進位表示為 01000001。

因為最終都是轉換成二進位序列形式，所以用何種方式轉換不會影響浮水印品質的評估，但不同的轉換方式會影響到轉換後的二進位序列長度，如果轉換後的長度過長，可能會沒辦法嵌入到圖片中。本研究選擇隨機產生 22 個 1 位元組的數字，其範圍為 [0, 255]，總長度為 176 位元。產生的結果：

```
[172,  10, 127, 140,  47, 170, 196, 151, 117, 166,
  22, 183, 192, 204,  33, 216,  67, 179,  78, 154, 251,  82]
```
。

（三）將資訊嵌入影像

欲嵌入的資訊準備妥當後，即可開始嵌入資訊。

首先將目標圖片的色彩空間由 RGB 轉換成 YUV。RGB 色彩空間是分別以 Red 紅色、Green 綠色、Blue 藍色，以光的三種顏色，表示三個通道，組合以表示單一像素。YUV 色彩空間是以 Y 通道表示 Luminance 明亮度，U、V 通道表示 Chrominance 色度。YUV 色彩空間的發明是因為人眼對於亮度較為敏感，因此適當的減少色度的採樣次數，可以有效地節省空間、頻寬。在隱藏浮水印上使用 YUV 色彩空間同樣也是利用人眼的特性，只對色度通道加上隱藏浮水印，不修改亮度值（Y通道），即可降低視覺上的差異。其轉換數學式為： [12]

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.299 & -0.587 & 0.886 \\ 0.701 & -0.587 & -0.114 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

提取轉換後的色度通道，分割成一個一個的正方形區塊（Block），每個區塊長寬為區塊大小（Block size）。對於一張長寬為 $H \times W$ 的目標圖片，最多可以嵌入 $\left(\left\lfloor H/_{Block\ size} \right\rfloor\right) \cdot \left(\left\lfloor W/_{Block\ size} \right\rfloor\right)$ 個位元。

對每一個區塊，使用 DCT 轉換，從空間域轉換到頻率域，DCT 在隱藏浮水印上，擁有著較高的強健性。對套用 DCT 後的區塊套用 SVD，將每個區塊分解成 U, D, V^T 三個矩陣。SVD 奇異值能讓圖片受到攻擊時，奇異值矩陣 D 中主要的數值 $\gamma_{1,1}, \gamma_{2,2}$ 不會出現大幅度的更動，以增加對於雜訊的容忍度。

過去的隱藏浮水印直接將原始資料轉換後，長度為 l 的二進位序列 $B = \{b_0, b_1, \dots, b_{l-1}\},\ B \in 0, 1$ ，依照順序直接加入區塊中。定義品質係數 Q_1, Q_2，對區塊編號 i 的奇異值矩陣 D 的第一、第二個奇異值 $\gamma_{1,1}, \gamma_{2,2}$ 進行修改： [13] [14] [15]

$$\gamma_{k,k} = Q_k\left[\left\lfloor \gamma_{k,k} \div Q_k \right\rfloor + 0.25 + 0.5 * M_{(i \bmod l)}\right], k \in 1, 2$$

然而，若圖片遭到裁切，且不知道原始圖片的尺寸及位置，就沒辦法知道還原出來的區塊是代表哪一個區塊。舉例說明如圖：

圖 3, 4、原始圖片與截圖，還原後的輸出示意圖。還原時輸入為原圖大小的情況下（左圖），可以從第一個位元（左圖的左上角）開始還原資訊。如果還原時輸入一塊截圖（右圖），且原始圖片起始點未知，找不到從何開始還原資訊，就無從得知目前還原出來的區塊，在原圖中是代表哪一部分的資訊。

因此，我基於一種稱為 Data matrix 的二維條碼進行構想，設計了簡單、適合用在隱藏浮水印的二維條碼，大小為 16 x 16 位元。定義如下表：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Fixed border		F	i	x	e	d		b	o	r	d	e	r			0
	Byte 01								CRC16 - 1				Block index			1
	Byte 02														F	2
	Byte 03								Byte 15				CRC16 - 2		i	3
	Byte 04														x	4
	Byte 05								Byte 16						e	5
	Byte 06														d	6
	Byte 07								Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22		7
	Byte 08														b	8
	Byte 09														o	9
	Byte 10														r	10
	Byte 11														d	11
	Byte 12														e	12
	Byte 13														r	13
	Byte 14															14
Fixed border																15

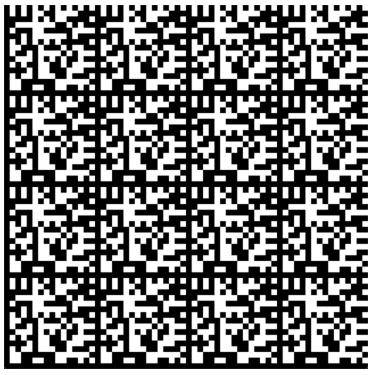
表 3、自製簡易條碼定義表。上方、右方 0 ~ 15 非條碼內容僅表示第 0 ~ 15 位元。

表中最上方一列與最右方一欄為索引（Index），並非條碼的一部分。條碼的最外層為固定的邊框（標註：Fixed border），用於定位還原後的位置。Byte 01 到 Byte 22 為該條碼能存放資料的區域，總共 22 位元組。右上角 4 位元的 Block index 則是用於標註該條碼的編號，當需要嵌入的資料的二進位序列長度超過 22 位元組時，可以用 Block index 標註該條碼為第幾個條碼，利用增加條碼的數量來換取更大的資訊容量。CRC16 – 1 和 CRC16 – 2 兩區域共計 16 位元，使用 CRC16 Modbus 協議。這是一種常用的循環冗餘校驗演算法，常用於檢查資料通訊中是否存在錯誤，在此用於檢驗條碼的正確性。

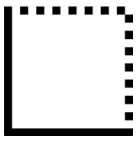
基於自行設計的條碼 $M_{i,j}$，定義品質係數 Q_1, Q_2，對於整張圖片的 $\text{Block}_{i,j}$ 的奇異值矩陣 D 的第一、第二個奇異值 $\gamma_{1,1}, \gamma_{2,2}$ 進行修改：

$$\gamma_{k,k} = Q_k\left[\left|\gamma_{k,k} \div Q_k\right| + 0.25 + 0.5 * M_{(i \bmod 16),(j \bmod 16)}\right], k \in 1, 2$$

即代表將條碼每個位元對每個區塊嵌入。若有超過 1 種條碼要嵌入，可將條碼從左到右、從上到下放置。只有一種條碼的成果如下圖：



截圖還原後的結果



固定邊框

定位後的條碼

圖 5, 6、原始圖片與截圖，分別還原後的輸出示意圖，使用自行設計的條碼。

圖 7,8、固定邊框示意圖及定位後的條碼示意圖。

還原時輸入為原圖大小的情況下（左圖），可以清楚的看見條碼被整齊的嵌入。

當還原時輸入一塊截圖（右圖），也可以從固定邊框（Fixed border）

定位條碼的起始位置，找出應該要從哪裡開始讀取資訊。

完成資訊的嵌入後，只要將所有區塊進行逆 SVD，再進行逆 DCT，將圖片還原到空間域，最後再覆蓋回原本的色度通道。將修改後的 YUV 色彩空間的圖片，轉換回 RGB 色彩空間，即為加上隱藏浮水印的圖片。轉換的數學式如下：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

整體嵌入浮水印的流程圖如下（以提取色彩通道 U 示範）：

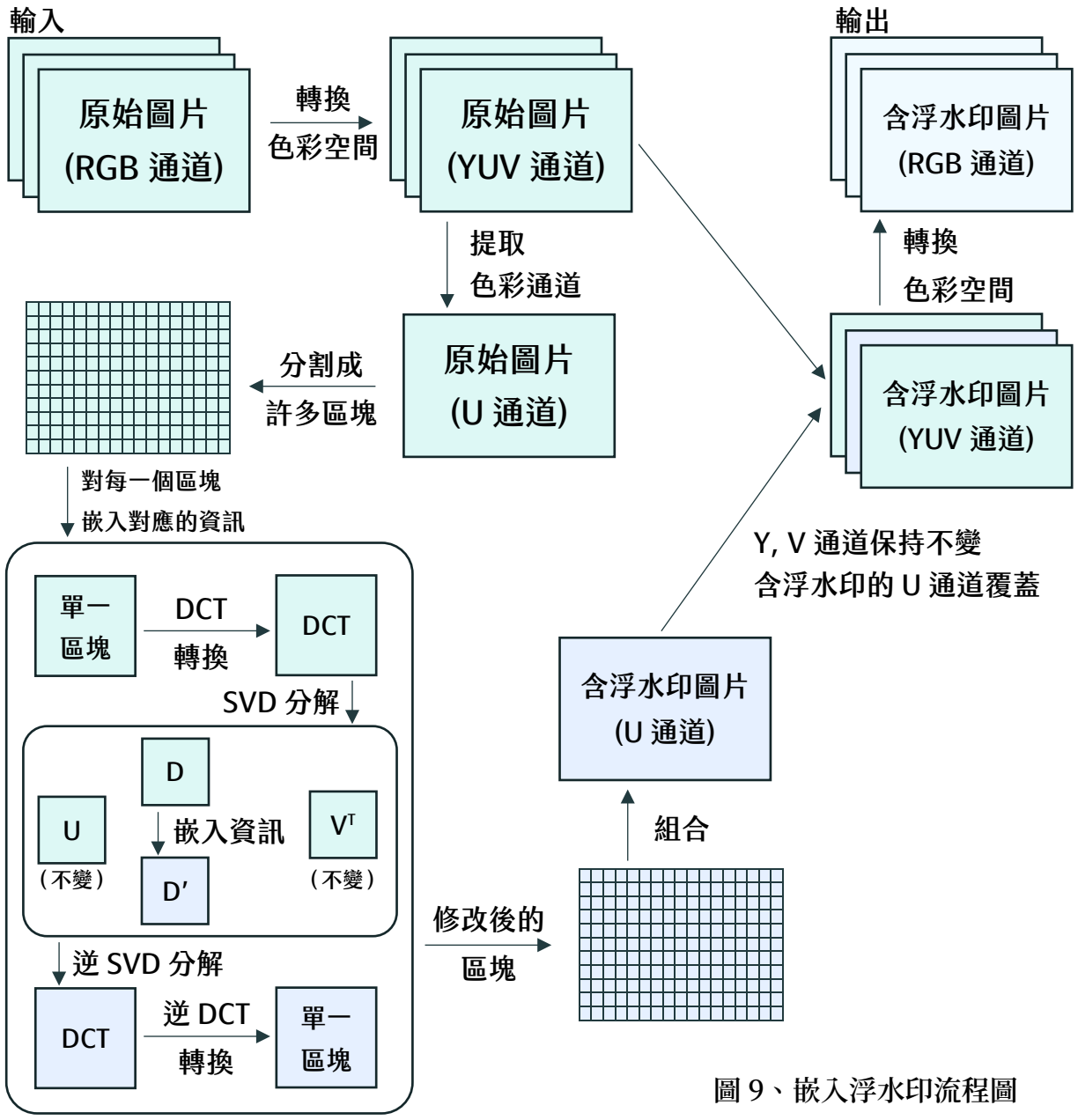
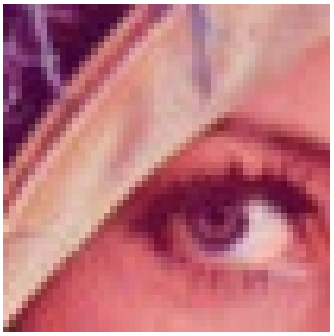
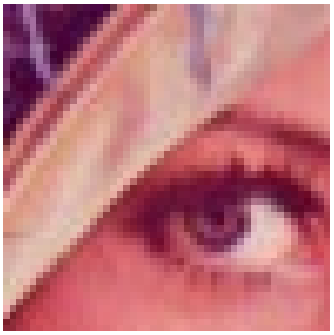
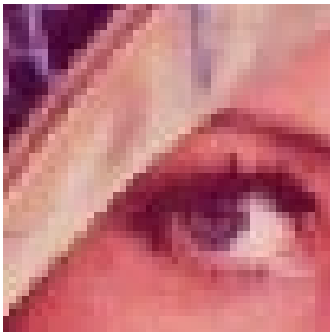
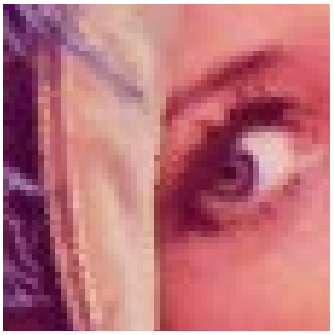


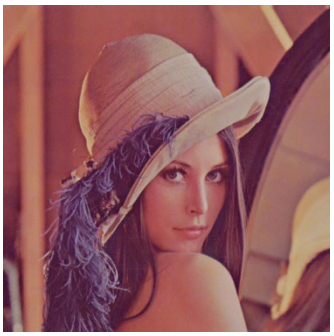
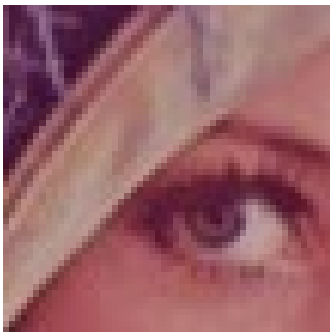
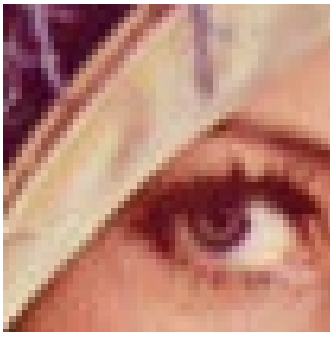
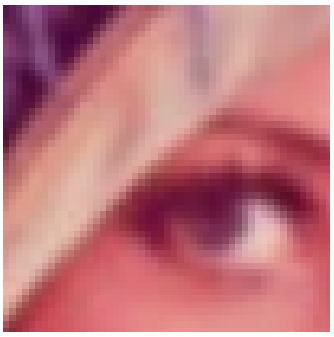
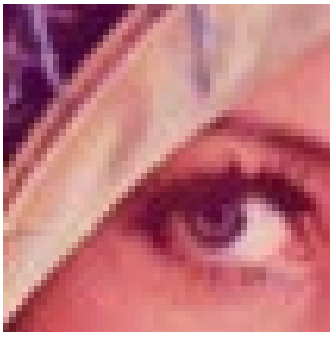
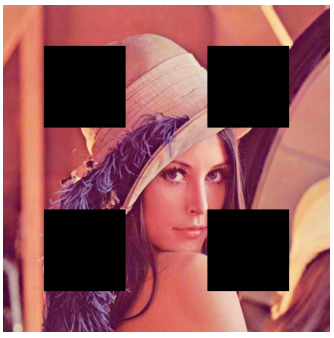
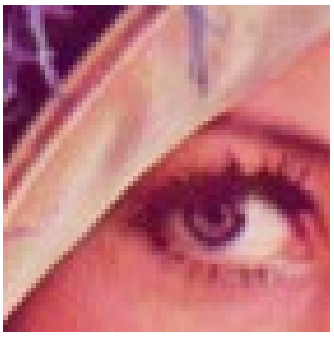
圖 9、嵌入浮水印流程圖

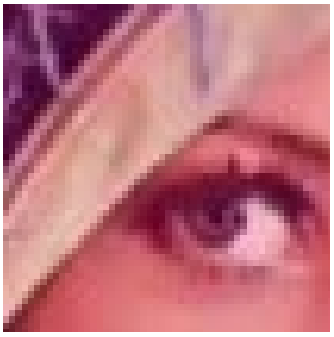
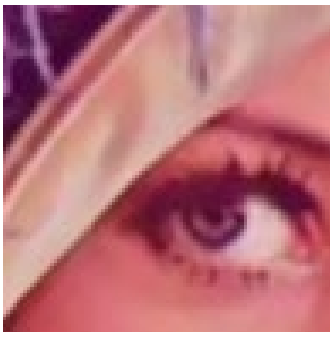
（四）生成模擬被攻擊的影像

不論是圖片或影片，在傳輸時，經常受到不同程度的壓縮，或遭到第三者修改。不同的攻擊手段都可能導致嵌入的隱藏浮水印，沒辦法解碼出正確結果。因此，本研究使用電腦視覺相關領域常用的 Lenna [16] 以及自行由手機拍攝的 8 張圖片作為測試圖片（詳見 P28 表 15）。對於每一張測試圖片，產生共 12 張圖片，其中 1 張不修改及 11 種模擬影像遭修改的圖片，以測試影像遭到攻擊時的情況（在此以 Lenna 作為示範被修改後的圖片）：

表 4、模擬被攻擊的影像表（共 3 面）

修改後的圖片示意圖	示意圖局部放大	項目名稱、描述
		**(1) 原始圖片** 未修改的含浮水印圖片 解析度 512 x 512 色彩深度 8bit
		**(2) 裁切** 中心 384 x 384 的區域之外加上黑色遮罩（像素值設為 0）
		**(3) 縮放** 將圖片降低為一半的解析度（512 x 512 → 256 x 256）
		**(4) 旋轉** 旋轉圖片 45 度（還原浮水印時會把圖片旋轉 -45 度復原，示意圖未復原）

		**(5) 亮度改變** 亮度及對比度降低 15%
		**(6) 色彩改變** 飽和度降低 10% 色相改變 +5%
		**(7) 高斯模糊** 模糊圖片 kernel size = 7 x 7 sigma = 2
		**(8) JPEG 壓縮** 使用 OpenCV 將圖片儲存成 .jpeg 格式，品質為 90（最大 100）。使用 JPEG 2000 標準。以模擬影片截取圖片後的壓縮。
		**(9) 覆蓋** 給圖片加上 4 個 128 x 128 的黑色遮罩（像素值設為 0） 遮罩區域占影像總面積的 1/4

		**(10) 椒鹽雜訊** 隨機令 2.5% 的像素為黑色（像素值設為 0），2.5% 的像素為白色（像素值設為 255） 評估時會固定隨機數種子
		**(11) H.264 編碼壓縮** 常見的主流影音平台都以 H.264 編碼為主，因此，本攻擊可以模擬影片加上浮水印之後，在網路的傳輸過程中受到的壓縮。
		**(12) AV1 編碼壓縮** AV1 是新的開源編碼格式，資料壓縮率相比 H.264 高約 50%，且免授權費，故越來越多影音平台開始採用 AV1 編碼。 [17]

以上編號 2~7 使用 Pytorch 的配套函式庫 Torchvision 完成，編號 9, 10 則是讀取照片後，直接修改目標像素值為 0 或 255。除了編號 8 之外，其餘的圖片皆儲存成 PNG 檔，為一種無損壓縮的影像格式。

編號 11, 12 由影片中擷取畫面作為測試。影片使用剪輯軟體，設定為 8bit 色深，解析度 512 x 512，每秒 30 幀。將含浮水印圖片顯示長度設定為 150 幀，前面 60 幀亮度由 0% 到 100%（淡入），後面 60 幀亮度由 100% 到 0%（淡出）。匯出 H.264 及 AV1 編碼的影片，位元速率（Bitrate）設定 1024kbps。擷取亮度為 100%的第 76 幀作為 H.264 及 AV1 編碼壓縮的測試圖片。**這兩項測試主要模擬影像在被編碼成影片之後，可能受到的畫質壓縮。若經過影片編碼壓縮後可以正常還原，代表本隱藏浮水印演算法放在動態影像中也能正常運作。**

（五）還原浮水印

還原浮水印的過程與嵌入浮水印類似，同樣需要將輸入的圖片由 RGB 轉換成 YUV 色彩空間。對當時嵌入浮水印色彩通道進行提取，分成數個正方形的區塊，對每一個區塊進行 DCT，接著進行 SVD，得到 U, D, V^T 三個矩陣。品質係數 Q_1, Q_2 與嵌入資料相同。

過去的隱藏浮水印還原時，須設定兩個變數，分別為與嵌入的資料相同長度 l 的二進位序列 $B = \{b_0, b_1, \dots, b_{l-1}\}$ 及紀錄區塊數量的陣列 $C = \{c_0, c_1, \dots, c_{l-1}\}$。品質係數 Q_1, Q_2 與嵌入資料相同，對區塊編號 i 的 D 矩陣的第一、第二個奇異值 d_1, d_2 進行讀取：

$$\begin{cases} 若 d_1 \ mod \ Q_1 < {Q_1}/{2}, 令 A_1 = 0, 否則 A_1 = 1 \\ 若 d_2 \ mod \ Q_2 < {Q_2}/{2}, 令 A_2 = 0, 否則 A_2 = 1 \end{cases}$$

$$b_{(i \ mod \ l)} = b_{(i \ mod \ l)} + 0.75 \times A_1 + 0.25 \times A_2, \qquad c_i = c_i + 1$$

對所有區塊進行完上述過程後，對序列 B 與陣列 C 進行以下操作：

$$若 {b_i}/{c_i} < 0.5, 令 b_i = 0, 否則 b_i = 1$$

最終得到輸出結果 $B = \{b_0, b_1, \dots, b_{l-1}\},\ B \in 0, 1$。

本研究的設計（條碼）還原時，輸入長寬為 $H \times W$ 的目標圖片，定義從每一個分割的區塊，還原的資料為一個長寬為$\left(\left\lfloor {H}/{Block\ size} \right\rfloor\right) \times \left(\left\lfloor {W}/{Block\ size} \right\rfloor\right)$ 的矩陣 M。還原資料時，對 $\text{Block}_{i,j}$ 的 D 矩陣的第一、第二個奇異值 d_1, d_2 進行讀取：

$$\begin{cases} 若 d_1 \ mod \ Q_1 < {Q_1}/{2}, 令 A_1 = 0, 否則 A_1 = 1 \\ 若 d_2 \ mod \ Q_2 < {Q_2}/{2}, 令 A_2 = 0, 否則 A_2 = 1 \end{cases}$$

$$M_{i,j} = 0.75 \times A_1 + 0.25 \times A_2$$

對所有 Block 進行完上述過程後，透過固定的條碼規律，定位每一格條碼。對每一個條碼，對其中的 data 1 ~ data 22 進行讀取，即為嵌入的資料。對 data 1 ~ data 22 計算 CRC16，並讀取條碼 CRC16，如果兩者不相同，則判定該條碼還原的資料是錯誤的。最後，如果當時嵌入了超過一個條碼，還需要讀取 Block index 以確認該條碼是第幾個條碼。

如果每一個的資料都錯誤，則 Block index 相同的所有條碼中的元素進行加總，形成一個新條碼 N。定義 Block index 相同的有 n 個條碼，進行以下操作，使形成的條碼 N 中只包含 0 和 1：

$$若 {}^{M_{i,j}}/_{n} < 0.5, 令 N_{i,j} = 0, 否則 N_{i,j} = 1$$

對條碼 N 的 data 1 ~ data 22 進行讀取，計算 CRC16，並讀取條碼 N 的 CRC16，如果兩者不相同，則判定該影像無法正確還原任何資料。

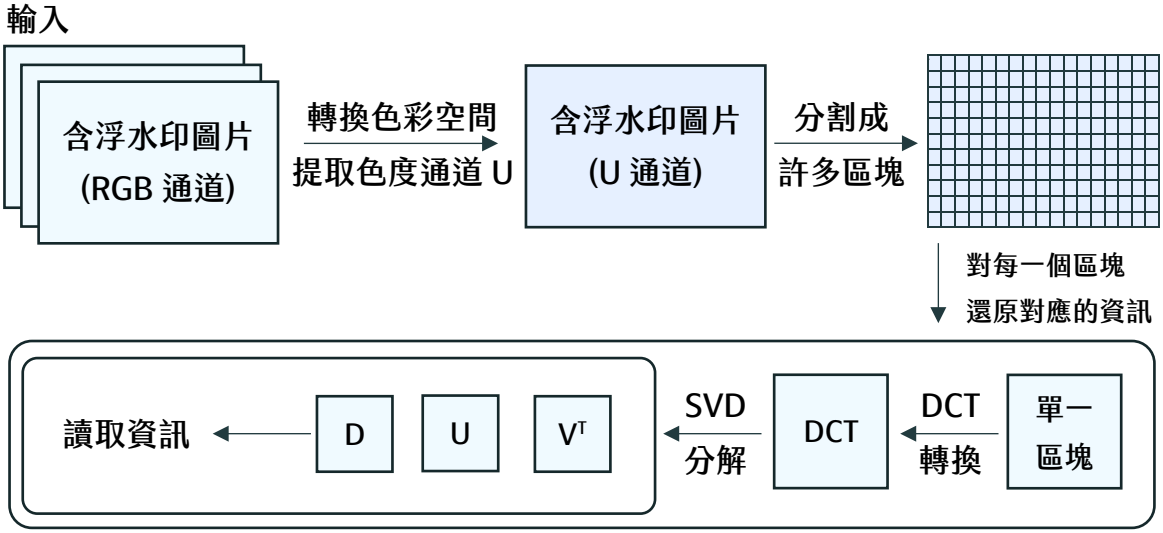


圖 10、還原浮水印流程圖（以提取色彩通道 U 示範）

（六）評估嵌入浮水印品質

1. 影像品質－峰值信噪比（Peak Signal to Noise Ratio, PSNR）

$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right) \qquad MSE = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I_1(i,j) - I_2(i,j)]^2}{m \times n}$$

對於一張色彩深度為 B 位元的圖片，$R = 2^B - 1$，本研究使用的圖片為 8 位元，故 $R = 255$ 。均方誤差（Mean Square Error, MSE）公式中，I_1 及 I_2 為比較的兩張圖片，大小為 $m \times n$。PSNR 的單位為分貝（dB），數值越高代表兩張圖片的像素之間的顏色更加相似，可以用來評估加上浮水印後會不會有明顯的色差。對於人眼來說，30dB 以下能看出明顯差異，40dB 以上即難以分辨出兩張圖片的差異。 [18]

2. 影像品質－結構相似度指標（Structural Similarity Index, SSIM）

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\beta}$$

$$\text{其中}\quad \begin{cases} l(x,y) = \dfrac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} & \textbf{比較亮度} \\ c(x,y) = \dfrac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} & \textbf{比較對比度} \\ s(x,y) = \dfrac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} & \textbf{比較結構} \end{cases}$$

$\mu_x,\ \mu_y$ 為 $x,\ y$ 的平均值（平均亮度），$\sigma_x,\ \sigma_y$ 為標準差，$\sigma_x\sigma_y$ 為相關係數，$C_1,\ C_2,\ C_3$ 為常數。本研究參照 MATLAB 中的 SSIM 函數，使用 $\alpha = \beta = \gamma = 1$, $C_1 = (0.01 \times 255)^2,\ C_2 = (0.03 \times 255)^2,\ C_1 = {C_2}/{2}$，此時 SSIM 可簡化成：

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

其範圍為 [0, 1] ，數值越大代表輸入的圖片 (x,y) 更相似，當 SSIM = 1 時，代表兩張圖片相同。 [19] [20] [21]

3. 浮水印效能－嵌入資訊的還原能力及正確率

對一張圖片的每一個區塊進行還原，比對該區塊嵌入時的資料。若還原後與嵌入時的資料相同，則「可還原資料圖片數」加 1。

4. 浮水印效能－正確率

定義 **正確率** = **正確數目** / **總區塊數目**，範圍為 [0, 1] 。越高的正確率代表能從嵌入浮水印的圖片中，正確地還原更多資訊，該浮水印的抗攻擊能力越好。

5. 浮水印效能－運算速度

在圖片讀取完成之後開始計時，直到嵌入浮水印完成，可以輸出圖片時，停止計時。單位為毫秒（ms）。

（七） 調整浮水印參數

設計好隱藏浮水印的嵌入及還原方式之後，在實際執行嵌入浮水印之前，有幾項可能的變因會導致結果不同。針對這些變因進行測試，並找出平衡影像品質、還原能力、運算速度的最佳的參數組合：

1. 資料的準備：直接嵌入資料（舊有方法），與嵌入自行設計的條碼對比。
2. 色彩空間：使用 YUV 色彩空間是否具有優勢、應該使用哪一個色彩通道。
3. 區塊大小：區塊大小對於還原浮水印的能力、影像品質、運算速度的影響。
4. 品質係數：品質係數 Q_1, Q_2 對於還原浮水印的能力、影像品質的影響。

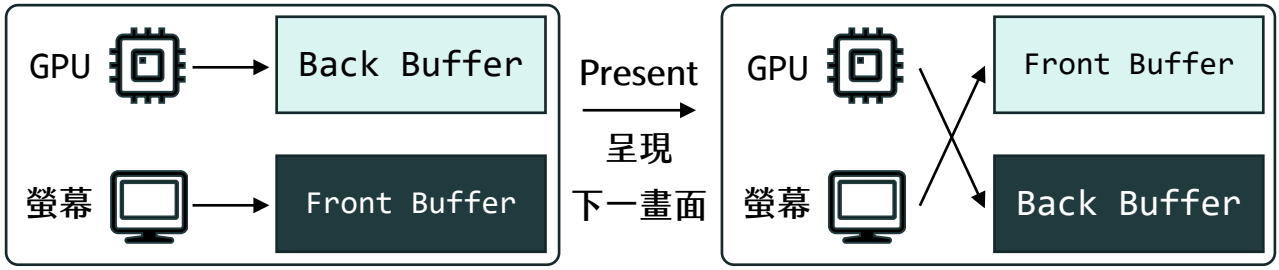
（八）「動態影像」隱藏浮水印的實際應用

1. 即時嵌入隱藏浮水印

完成主要演算法的設計之後，將演算法套用在「動態影像」上，以驗證該演算法對於即時在動態影像添加浮水印的可行性。撰寫一個電腦程式，捕捉特定視窗，並即時輸出含隱藏浮水印的畫面。為了方便以人眼對比嵌入浮水印前後的畫面，本程式新建一個視窗顯示含隱藏浮水印的畫面。

本研究參考微軟的螢幕捕捉使用範例 [22]，使用 C++ 撰寫程式捕捉畫面。呼叫 Windows.Graphics.Capture API 使程式能夠捕捉特定視窗的畫面，並且存放到 Direct3D 11 Swap Chain （交換鏈）的 Back Buffer（後緩衝區）中。Direct3D 11 是一個用於 Windows 平台的圖形 API，其中 Swap Chain 是與影像顯示高度相關的一個概念。Swap Chain 中通常具有多個 Buffer，Front Buffer（前緩衝區）是螢幕正在顯示的內容，而 Back Buffer 則是 GPU（圖形處理器）完成渲染（Render）的畫面。當新的一個畫面 Present（呈現）時，Swap Chain 交換兩個 Buffer 的指標，兩者之間「**功能**」互換，「**內容**」不變。

圖 11、交換鏈概念圖



欲添加隱藏浮水印，只要讀取 Back Buffer 中的內容，套用隱藏浮水印演算法，再寫入到 Back Buffer 中，最後呼叫 Present，讓本程式顯示已經加上浮水印的畫面，即完成即時加入浮水印的過程。

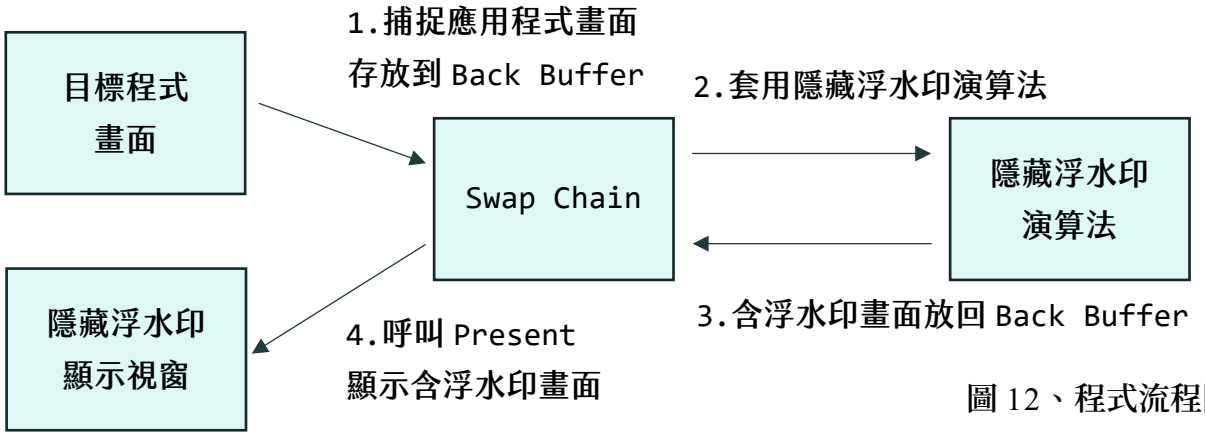


圖 12、程式流程圖

為了評估可行性，設定程式擷取影片撥放軟體 PotPlayer，播放以解析度 2560 x 1440，120 FPS 錄製的遊戲影片。測量程式每幀畫面輸出的平均間隔（幀生成時間，單位：毫秒 ms），及嵌入浮水印消耗的平均時間，用以評估程式執行速度。每秒幀數（Frames Per Second, FPS）= 1000 / 幀生成時間（ms）。常見的電影為 24 FPS，電腦螢幕為 60 FPS。幀生成時間越低越好，FPS 越高越好，代表運算速度越快、畫面越流暢。

2. 隱藏浮水印還原視覺化

除了從影像中提取資訊之外，此演算法也可以應用於偵測畫面是否遭到人為修改，在實際應用中，可以應用於法院，以證明提供的影片是原始檔案，未經任何修改，是有效的證物。

在動態影像嵌入浮水印的過程中，儲存嵌入時的 Timestamp（時間戳）：使用條碼中可嵌入的資訊的 22 位元組其中的 6 位元組，以 4 位元組儲存整數部分（秒），以 2 位元組儲存小數部分（毫秒），剩下 16 位元組可以使用者需求儲存其他資訊。製作一個程式，嘗試還原影片每個畫面的浮水印，並進行以下判斷：

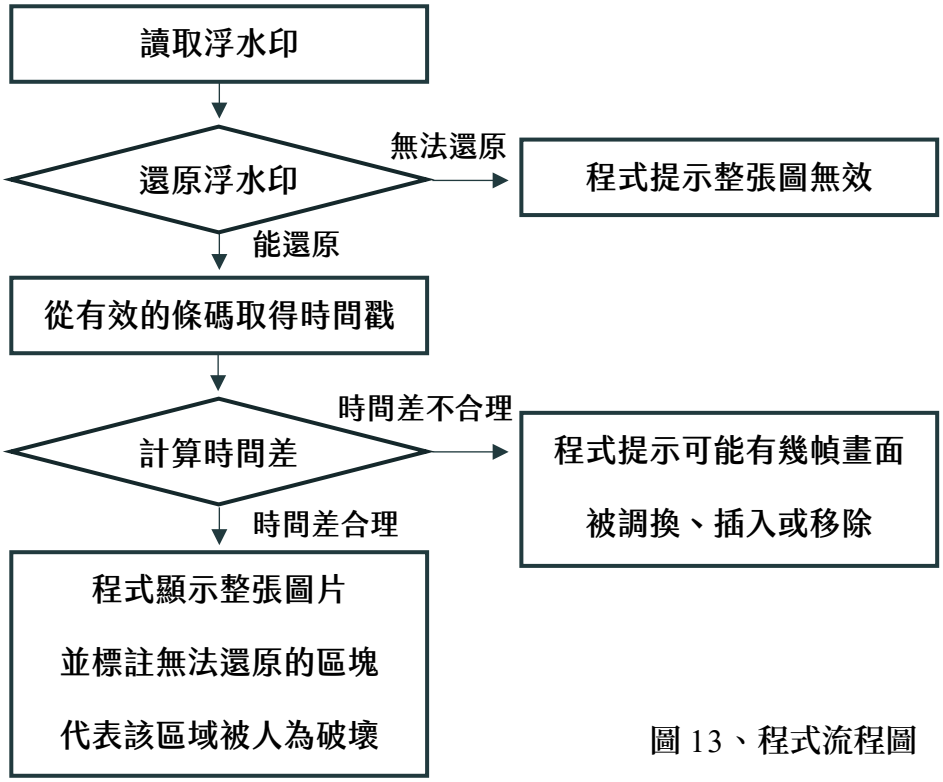


圖 13、程式流程圖

其中，合理的時間差應該與上一幀差距約在影片的平均幀生成時間的 0.5 ~ 2.5 倍（考慮嵌入浮水印可能有延遲），如 60 FPS 的影片，幀生成約為 16.6 毫秒，即兩幀畫面之間，時間戳的差應該約在 8 ~ 41 毫秒之間。

肆、研究結果與分析

（一） 資料的準備：直接嵌入資料（舊有方法），與嵌入自行設計的條碼對比

表 5、直接嵌入資料和嵌入自行設計與抗攻擊能力、影像品質的關係表。

變因	**直接嵌入資料**		**嵌入條碼**	
攻擊類型	**正確率 / 可還原資料圖片數↑**			
未攻擊	1.00	9	1.00	9
裁切	0.78	9	0.81	9
縮放	0.95	9	0.96	9
旋轉	0.91	9	0.92	9
亮度改變	0.58	2	0.52	**3**
色彩改變	0.68	4	0.67	**6**
高斯模糊	0.90	3	0.91	5
JPEG 壓縮	0.96	9	0.97	9
覆蓋	0.88	9	0.89	9
椒鹽雜訊	0.93	9	0.89	9
H.264 壓縮	0.77	0	0.85	**3**
AV1 壓縮	0.81	2	0.88	**4**
耗時(ms) ↓	79.1		79.5	
含浮水印的未攻擊影像與原始影像的比較				
SSIM↑	0.93		0.92	
PSNR (dB) ↑	34.74		35.23	

使用 U 色彩通道, Block Size = 8, $Q_1 = Q_2 = 50$

本項實驗主要目的為評估自行設計的條碼，除了在影像遇到裁切、遮罩等攻擊時，能輔助定位起始點，是否在其它攻擊方式中，也具有優勢。在本項實驗中，並沒有將圖像的原始位置設為未知，因此過去的嵌入方式遇到裁切、遮罩時也能正常還原出資訊。若將圖像的原始未知，過去的嵌入方式產生的結果都會因無法定位資料起始點而無法還原，這項實驗的意義就不大了。

自行設計的條碼，相較於過去的嵌入方式，在每一個區塊的準確度沒有太大差異，因為設計的條碼只改變了資料的排列順序，並沒有改變單一個位元在嵌入單一個區塊的嵌入方式。然而，由於 CRC16 校驗演算法的加入，可以對每一個條碼的資料與其中的 CRC16 比對，如果正確就可以說這個條碼是有效的。因此，在亮度、顏色改變時，以及面對影片編碼壓縮，以自行設計的條碼嵌入，還原時有較高的成功率。

（二） 使用 YUV 色彩空間是否具有優勢、應該使用哪一個色彩通道

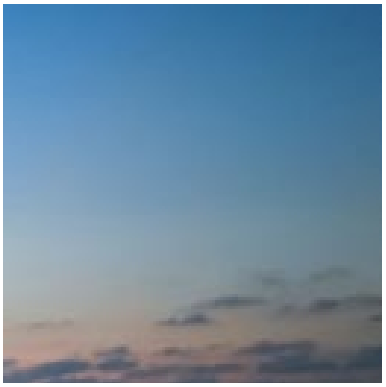
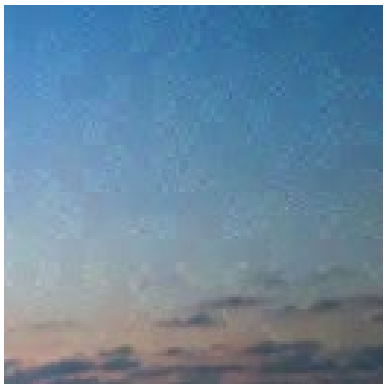
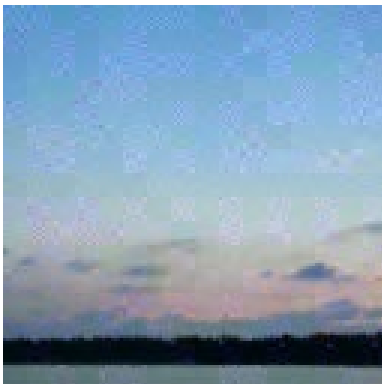
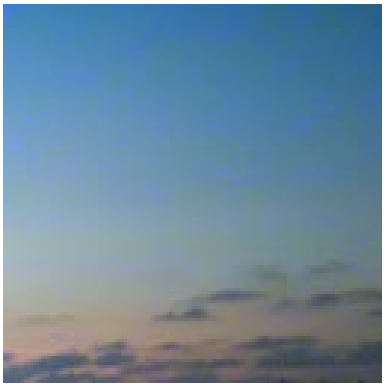
表 6、色彩空間和通道與抗攻擊能力、影像品質的關係表

變因	RGB		YUV		Y		U		UV	
攻擊類型	**正確率 / 可還原資料圖片數↑**									
未攻擊	0.99	9	1.00	9	1.00	9	1.00	9	1.00	9
裁切	0.79	1	0.80	9	0.83	9	0.81	9	0.80	9
縮放	0.69	5	0.81	5	0.73	3	0.96	9	0.95	9
旋轉	0.80	9	0.86	9	0.81	3	0.92	9	0.91	9
亮度改變	0.45	0	0.50	0	0.50	0	0.52	**3**	0.55	2
色彩改變	0.57	0	0.56	1	0.64	1	0.67	**6**	0.60	3
高斯模糊	0.66	1	0.80	5	0.66	0	0.91	5	0.89	3
JPEG 壓縮	0.85	5	0.96	9	1.00	8	0.97	9	0.95	7
覆蓋	0.87	9	0.89	9	0.89	8	0.89	9	0.88	9
椒鹽雜訊	0.54	0	0.67	0	0.52	0	0.89	9	0.93	9
H.264 壓縮	0.43	0	0.53	0	0.33	0	0.85	**3**	0.77	1
AV1 壓縮	0.45	0	0.64	0	0.51	0	0.88	**4**	0.81	2
耗時(ms) ↓	242.8		238.7		79.6		79.5		165.0	
含浮水印的未攻擊影像與原始影像的比較										
SSIM↑	0.94		0.90		0.90		0.92		0.93	
PSNR (dB) ↑	30.79		30.45		35.81		35.23		34.23	

使用自行設計的條碼，Block Size = 8, $Q_1 = Q_2 = 50$

本項實驗測試了在不同色彩空間和通道嵌入浮水印的成效。經過測試，對於單一色度通道嵌入浮水印有最佳還原能力、低運算時間以及更高的影像品質。值得注意的是，對 Y 通道嵌入浮水印的 PSNR 最高，但實際上人眼看起來 Y 通道的嵌入痕跡比 U 通道明顯許多，其對比圖如下：

表 7、不同通道嵌入浮水印對比圖

 原始圖像	 原始圖像（局部放大）
 RGB 通道嵌入浮水印	 YUV 通道嵌入浮水印
 Y 通道嵌入浮水印	 U 通道嵌入浮水印

（三） 區塊大小對於還原浮水印的能力、影像品質、運算速度的影響

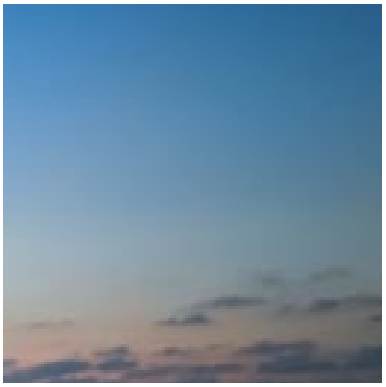
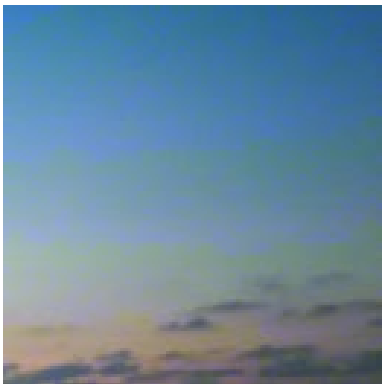
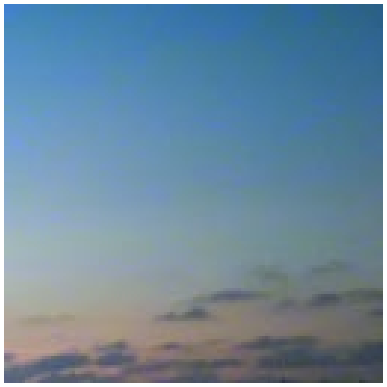
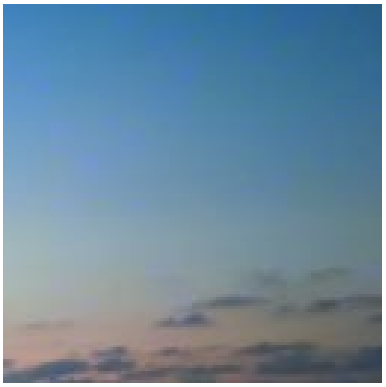
表 8、區塊大小與還原浮水印的能力、影像品質、運算速度的關係表

變因	2		4		8		16	
攻擊類型	**正確率 / 可還原資料圖片數↑**							
原圖	1.00	9	1.00	9	1.00	9	1.00	9
裁切	0.81	9	0.81	9	0.81	9	0.76	0
縮放	0.82	0	0.95	8	0.96	9	0.84	6
旋轉	0.85	0	0.93	9	0.92	9	0.86	6
亮度改變	0.87	9	0.71	6	0.52	3	0.52	0
色彩改變	0.84	9	0.74	6	0.67	**6**	0.59	0
高斯模糊	0.71	0	0.79	0	0.91	5	0.76	0
JPEG 壓縮	0.83	0	0.90	0	0.97	**9**	0.83	0
覆蓋	0.89	9	0.89	8	0.89	9	0.87	0
椒鹽雜訊	0.96	9	0.97	9	0.89	9	0.74	3
H.264 壓縮	0.75	0	0.80	1	0.85	**3**	0.71	0
AV1 壓縮	0.80	0	0.85	1	0.88	**4**	0.66	0
耗時(ms) ↓	983.1		265.2		79.5		48.7	
含浮水印的未攻擊影像與原始影像的比較								
SSIM↑	0.77		0.89		0.92		0.99	
PSNR (dB) ↑	25.45		31.63		35.23		41.60	

使用 U 色彩通道，使用自行設計的條碼， $Q_1 = Q_2 = 50$

本項實驗測試了在不同區塊大小嵌入浮水印的成效。區塊大小增大，有助於提升影像品質，但抵抗攻擊能力變差；區塊大小縮小，有更佳的抗攻擊能力，但影像品質快速降低，甚至人眼明顯可見，且運算時間增加。綜合評估之後，區塊大小 8 有最佳成效。局部放大對比圖如下：

表 9、不同區塊大小嵌入浮水印對比圖

 原始圖像	 原始圖像（局部放大）
 區塊大小 2	 區塊大小 4
 區塊大小 8	 區塊大小 16

（四） 品質係數 Q_1, Q_2 對於還原浮水印的能力、影像品質的影響

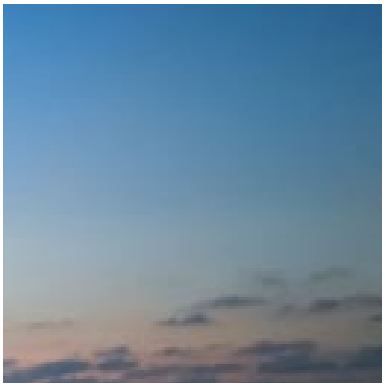
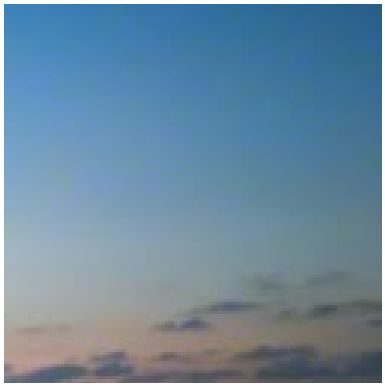
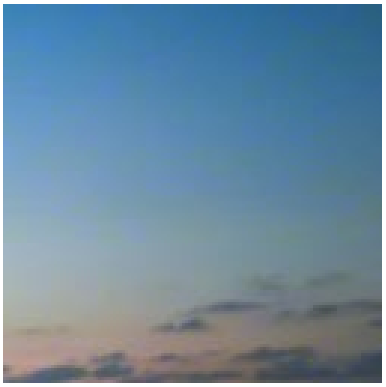
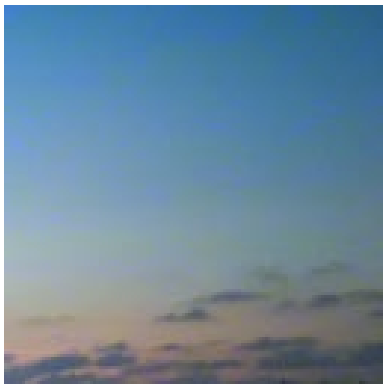
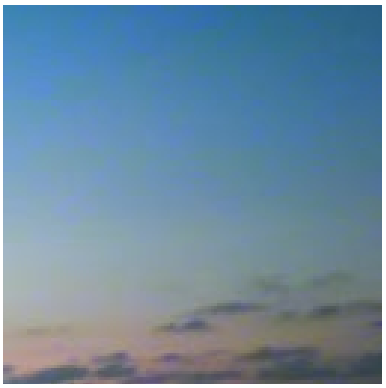
表 10、品質係數 Q_1, Q_2與還原浮水印的能力、影像品質的關係表

變因	20, 20		30, 30		40, 40		50, 50		60, 60	
攻擊類型	**正確率 / 可還原資料圖片數↑**									
原圖	1.00	9	1.00	9	1.00	9	1.00	9	1.00	9
裁切	0.80	0	0.81	9	0.76	9	0.81	9	0.81	9
縮放	0.78	0	0.84	7	0.92	9	0.96	9	0.98	9
旋轉	0.84	0	0.88	9	0.90	9	0.92	9	0.92	9
亮度改變	0.53	0	0.51	0	0.54	1	0.52	3	0.55	3
色彩改變	0.58	0	0.63	2	0.69	2	0.67	6	0.74	**7**
高斯模糊	0.67	0	0.73	0	0.85	0	0.91	5	0.93	5
JPEG 壓縮	0.71	0	0.81	0	0.90	4	0.97	9	0.99	9
覆蓋	0.89	9	0.89	9	0.86	9	0.89	9	0.89	9
椒鹽雜訊	0.74	1	0.82	4	0.89	7	0.89	9	0.92	9
H.264 壓縮	0.62	0	0.64	0	0.72	2	0.85	3	0.90	**4**
AV1 壓縮	0.59	0	0.63	0	0.69	1	0.88	4	0.92	**6**
含浮水印的未攻擊影像與原始影像的比較										
SSIM↑	0.99		0.98		0.97		0.92		0.94	
PSNR (dB) ↑	43.30		40.78		38.76		35.23		36.9	

使用 U 色彩通道，使用自行設計的條碼，Block Size = 8

本項實驗測試了不同品質系數嵌入浮水印的成效。品質係數過低時，抵抗攻擊能力極差。提升品質係數有助於增加抗攻擊能力，但影像品質會下降。整體約在$Q_1 = Q_2 = 50$時有足夠的抗攻擊能力，$Q_1 = Q_2 = 60$時對於影片編碼壓縮抗攻擊能力更高，其他攻擊的抵抗能力則沒有太大改變。

表 11、不同品質係數嵌入浮水印對比圖

 原始圖像	 原始圖像（局部放大）
 $Q_1 = Q_2 = 30$	 $Q_1 = Q_2 = 40$
 $Q_1 = Q_2 = 50$	 $Q_1 = Q_2 = 60$

（五） 即時影像隱藏浮水印的可行性評估

表 12、隱藏浮水印輸出結果比較

比較的兩張圖片	SSIM ↑	PSNR(dB) ↑
OpenCV-Python 與 OpenCV-C++	1.00	80

上述實驗是為了測試即時影像隱藏浮水印的程式使用的 OpenCV-C++的輸出與前述模擬攻擊、評估參數時的 OpenCV-Python 輸出是否不同。兩張圖片相同時，SSIM = 1 且 PSNR = 80。說明 OpenCV-Python 與 OpenCV-C++的輸出相同，前述實驗測試的演算法抗攻擊能力，在即時影像隱藏浮水印的程式也適用。

表 13、即時影像隱藏浮水印速度測試

變因	平均幀生成時間↓	平均 FPS↑	平均浮水印嵌入時間↓
不嵌入浮水印	18.22 ms	54.88 幀	0 ms
嵌入浮水印	18.38 ms	54.41 幀	2.17 ms

從測試的資料可以看出，發現就算不嵌入浮水印，幀生成時間（Frame Time）仍大致相同。說明程式執行瓶頸不在嵌入浮水印，而是在畫面捕捉的過程，導致幀生成時間長。若不考慮畫面捕捉的時間，僅考慮對影像嵌入浮水印，可達約 450FPS，高於一般使用者 60FPS 的需求，也就是即時對影像嵌入浮水印效能上是可行的。

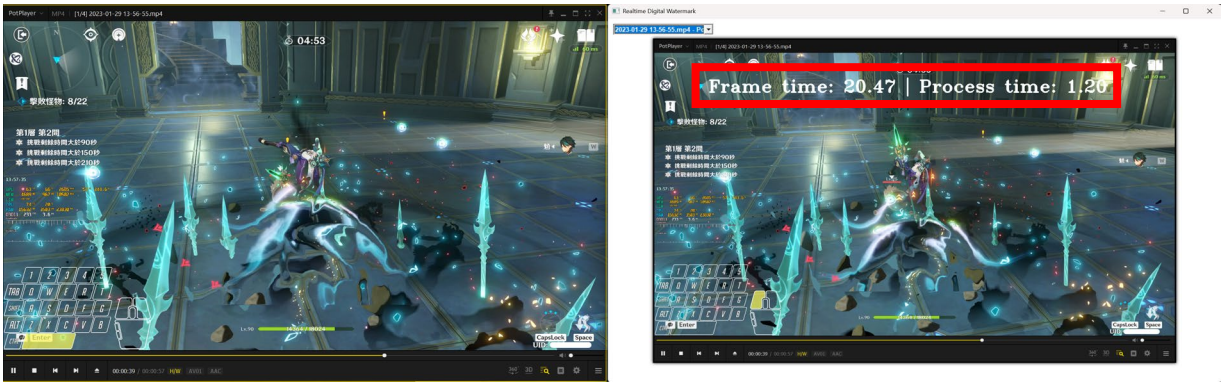
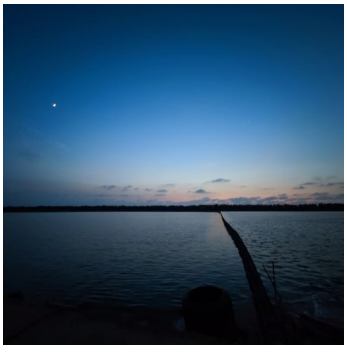
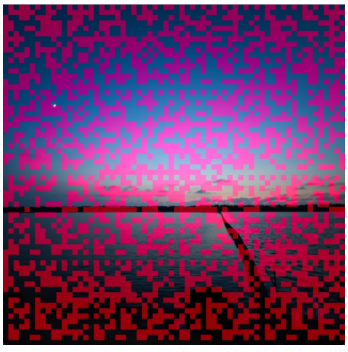


圖 15、程式輸出效果圖。左方為被捕捉畫面的視窗，右方為含浮水印的畫面。紅色方框中為幀生成時間。數據不同是因截圖時需要縮小視窗，解析度不同導致。

（六） 隱藏浮水印還原結果視覺化

因為作品說明書無法放影片，在此以圖片示範。前三個項目以「灰色天空」圖片為範例，而最後一項「偵測調換畫面」時，程式先讀取灰色天空圖片，再讀取「藍色天空」圖片，以模擬影片中某一畫面被替換的情形。

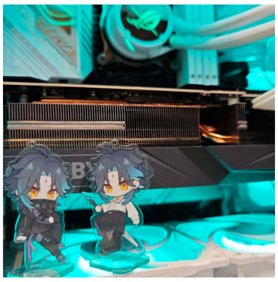
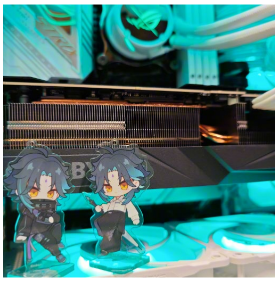
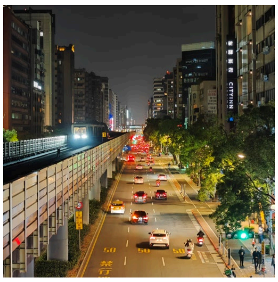
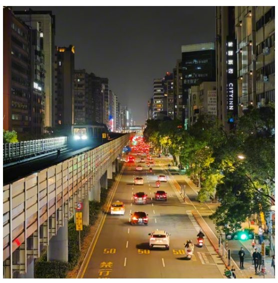
表 14、隱藏浮水印輸出結果比較

輸入圖片	錯誤區塊（紅色）	說明
縮放		
		為了節省空間，可能將影片的解析度稍微降低，沒有做其他修改。因此程式無標註錯誤區塊。
影片編碼壓縮		
		影片編碼是儲存成影片時常遇到的修改，程式無大範圍標註錯誤區塊。由於演算法沒辦法百分之百抵抗影片壓縮，部分區塊被錯誤的標註。
調整圖片的色彩（上半部分）		
		上半部分色相改變 20%，亮度、對比度、飽和度降低 20%。透過調色來掩蓋原本圖片的細節，是人為修改，程式標註錯誤區塊。
調換畫面		
		Prev: 1685899398.196 Expected: [.196, .200] Got: 1685898645.574 >> Incorrect timestamp 不同時間嵌入但可還原浮水印的兩張圖片，程式偵測時間戳無效。

（七） 隱藏浮水印影像品質展示

綜合以上實驗，得出對於動態影像最好的參數組合：對色度通道嵌入浮水印，Block size = 8, $Q_1 = Q_2 = 60$，使用自行設計的條碼作為資料嵌入方式。

表 15、隱藏浮水印結果（共 9 組圖片）

原始圖片			
含浮水印			
原始圖片			
含浮水印			
原始圖片			
含浮水印			

伍、結論與未來展望

一、 現有的影像隱藏浮水印多數是針對靜態影像進行的，尚未有將隱藏浮水印應用在動態影像上的範例，也沒有對動態影像編碼攻擊進行測試。本研究成功發展出基於 DCT 與 SVD 演算法，針對動態影像的即時隱藏浮水印系統。

二、 模擬影像被人為修改，評估隱藏浮水印還原能力、影像品質、運算速度，以證明演算法在找出最佳參數後，可使影像受到不同程度的干擾時，能正確地還原嵌入的資訊。

三、 透過修改資訊嵌入的方式，設計一套條碼，解決過去隱藏浮水印需要預先知道原始圖像位置，無法直接還原的問題，可增加隱藏浮水印還原能力。並且，使用者可更改條碼的定義，未知該定義的第三者無法透過還原出有效資訊，具保護資料安全的作用。

四、 本研究製作了兩個程式用以示範本技術的實際應用：

（一） 即時嵌入隱藏浮水印：能有效嵌入浮水印且執行速度能滿足需求。並且，該程式使用 Direct3D 11 API 捕捉及返回畫面，可以輕易地整合進各種會用到該 API 的程式，如影音撥放軟體、遊戲等，具有更多、更廣的應用價值，可以用來保護具有著作權、智慧財產權的內容。

（二） 隱藏浮水印還原結果視覺化：在動態影像（如監視器）錄製時，嵌入含每幀畫面時間戳的浮水印。當此程式還原浮水印時，偵測出某些區塊的浮水印遭破壞，代表該區域被竄改；或是時間戳與上一幀差距過大，代表該幀可能是額外插入的。而單純的影像縮放（如降低解析度以節省空間），浮水印能正常還原，不會誤報影像遭修改。此技術在法律上或許可以幫助證明該影片是有效的證物，防止有心人士利用影片編輯技術修改影像、製造偽證。

五、 未來展望

該演算法面對動態影像編碼壓縮攻擊，部分情況下不能還原出浮水印，仍須犧牲部分影像品質。希望未來能再結合其他頻率域－空間域轉換方式，針對動態影像編碼壓縮攻擊測試，找出適用於動態影像、抗攻擊能力與影像品質高的隱藏浮水印演算法。另外，在「隱藏浮水印還原結果視覺化」中，發現影片編碼壓縮會讓條碼特定區塊被破壞，導致整個條碼無效，因此可嘗試使用 Reed-Solomon 糾錯或其他糾錯演算法，使部分區域被破壞時也可以還原出全部正確資訊，增強抗攻擊能力。

陸、 參考文獻資料

[1] KTinker, "《暗黑破壞神 4》Alpha 內部測試版遊戲畫面外流，曝光創角與難度選項," 9 8 2022. [Online]. Available: https://today.line.me/tw/v2/amp/article/OpZeJe6. [Accessed 19 3 2023].

[2] Rocky, "《俠盜獵車手 6》高達 90 多部早期開發影片遭駭客大量洩漏，這應該是遊戲歷史上最大洩密事件之一," 19 9 2022. [Online]. Available: https://www.kocpc.com.tw/archives/460111. [Accessed 19 3 2023].

[3] KKVincent, "Take-Two 執行長稱《俠盜獵車手 6》外流事件對商業無影響，但對開發者造成了「情緒傷害」," 11 2 2023. [Online]. Available: https://www.kocpc.com.tw/archives/480036. [Accessed 19 3 2023].

[4] [deleted], "r/xboxone - One way MS tracked NDA breakers in the 360 era," 10 12 2018. [Online]. Available: https://old.reddit.com/r/xboxone/comments/a4y3n8/one_way_ms_tracked_nda_breakers_in_the_360_era/. [Accessed 14 3 2023].

[5] Shweta Wadhera, Deepa Kamra, Ankit Rajpal, Aruna Jain, Vishal Jain, "A Comprehensive Review on Digital Image Watermarking," in *WCNC-2021: Workshop on Computer Networks & Communications*, 2021.

[6] P. B. Dasgupta, "Algorithmic Analysis of Invisible Video Watermarking using LSB Encoding over a Client-Server Framework," *International Journal of Computer Trends and Technology (IJCTT),* vol. 36, no. 3, 2016.

[7] 辜慶軒, 胡懷祖, 徐鈴淵, "於 DWT-SVD-DCT 轉換域所發展之影像浮水印技術," *International Journal of Advanced Information Technologies(IJAIT),* vol. 8, no. 1, 2014.

[8] N. Ahmed, T. Natarajan, K. R. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers,* vol. 100(1), pp. 90-93, 1974.

[9] Chin-Chen Chang, Piyu Tsai, Chia-Chen Lin,, "SVD-based digital image watermarking scheme," *Pattern Recognition Letters,* vol. 26, no. 10, pp. 1577-1586, 2005.

[10] Begum, Mahbuba, Mohammad Shorif Uddin, "Analysis of digital image watermarking techniques through hybrid methods," *Advances in Multimedia,* vol. 2020, pp. 1-12, 2020.

[11] Rahman, Md. Maklachur, "A DWT, DCT and SVD Based Watermarking Technique to Protect the Image Piracy," *International Journal of Managing Public Sector Information and Communication Technologies,* vol. 4, 2013.

[12] "RGB-YUV Conversion," [Online]. Available: https://www.cs.sfu.ca/mmbook/programming_assignments/additional_notes/rgb_yuv_note/RGB-YUV.pdf. [Accessed 19 3 2023].

[13] @guofei9987, "GitHub - guofei9987/blind_watermark," 3 12 2022. [Online]. Available: https://github.com/guofei9987/blind_watermark/tree/839af4c7850db1921108d30cf4dc92c8e8ebb130. [Accessed 27 2 2022].

[14] @lishuaijuly, "GitHub - lishuaijuly/watermark," 5 11 2017. [Online]. Available: https://github.com/lishuaijuly/watermark/tree/23d167192aa90dcb3293448174211eead1367d38. [Accessed 27 2 2023].

[15] @ShieldMnt, "GitHub - ShieldMnt/invisible-watermark," 16 11 2022. [Online]. Available: https://github.com/ShieldMnt/invisible-watermark/tree/0690c09a9ec6e665b5072aa11deea12467e47afe. [Accessed 27 2 2023].

[16] "The Rest of the Lenna Story," 3 11 2001. [Online]. Available: http://www.lenna.org/. [Accessed 28 2 2023].

[17] Y. Liu, "AV1 beats x264 and libvpx-vp9 in practical use case," 10 4 2018. [Online]. Available: https://engineering.fb.com/2018/04/10/video-engineering/av1-beats-x264-and-libvpx-vp9-in-practical-use-case/. [Accessed 19 3 2023].

[18] Alain Horé, Djemel Ziou, "Image quality metrics: PSNR vs. SSIM," in *ICPR 2010*, 2010.

[19] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, Eero P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing,* vol. 13, no. 4, pp. 600-612, APRIL 2004.

[20] @xinntao, "BasicSR/basicsr/metrics・XPixelGroup/BasicSR," 8 9 2022. [Online]. Available: https://github.com/XPixelGroup/BasicSR/tree/b0ee3c8/basicsr/metrics#psnr-%E5%92%8C-ssim. [Accessed 27 2 2023].

[21] I. The MathWorks, "Structural similarity (SSIM) index for measuring image quality," 2021. [Online]. Available: https://www.mathworks.com/help/images/ref/ssim.html. [Accessed 27 2 2022].

[22] @ajbennet, @robmikh, "Windows.UI.Composition-Win32-Samples/cpp/ScreenCaptureforHWND," 10 1 2023. [Online]. Available: https://github.com/Microsoft/Windows.UI.Composition-Win32-Samples/tree/830f854/cpp/ScreenCaptureforHWND. [Accessed 28 2 2023].

【評語】052514

1. 此作品透過 DCT-SVD 演算法，即時對動態影像嵌入人眼幾乎不可見的浮水印。透過少量的實驗資料顯示，演算法具有一定的抗攻擊性，可以抵抗影像受到壓縮、裁剪等基本的浮水印攻擊。

2. 浮水印的研究較不具創新性，建議要加強應用情境的說明，具以說明所發展的技術的適用性。

3. 建議可多做一些實驗，以驗證演算法的可靠性。另也可與其他浮水印技術做一比較。

作品海報

摘要

本研究透過 DCT-SVD 演算法，即時對動態影像，嵌入人眼幾乎不可見的浮水印。浮水印有多種用途，例如版權方可在動態影像中，嵌入被授權者的資訊。當被授權者私自將影像給予未授權者，版權方可以透過還原浮水印，追蹤違反授權的被授權者，並採取必要的法律行動，以保障版權方的權益。

經過測試，此演算法具有一定的抗攻擊性，可以保障影像若受到他人壓縮、裁剪等操作後，能還原嵌入的浮水印資料，且運算速度快，可用於即時動態影像。

研究動機

不時能看到新聞報導，某遊戲測試版本因畫面洩漏，對遊戲公司造成了各種損失。遊戲公司開發一款遊戲需要耗費大量資源，在遊戲正式發布前會先進行內部測試，以驗證玩法是否符合玩家需求。為了解決可能的洩漏問題，許多遊戲公司會用肉眼可見的浮水印追蹤洩漏者，但這種方式會影響視覺觀感。

因此，本研究在測試，是否有可能在動態影像中嵌入隱藏浮水印，以保障版權方的權益，且將視覺觀感的影響最小化，並探討、實作此技術的其他應用。

研究設備與軟體

1 軟體

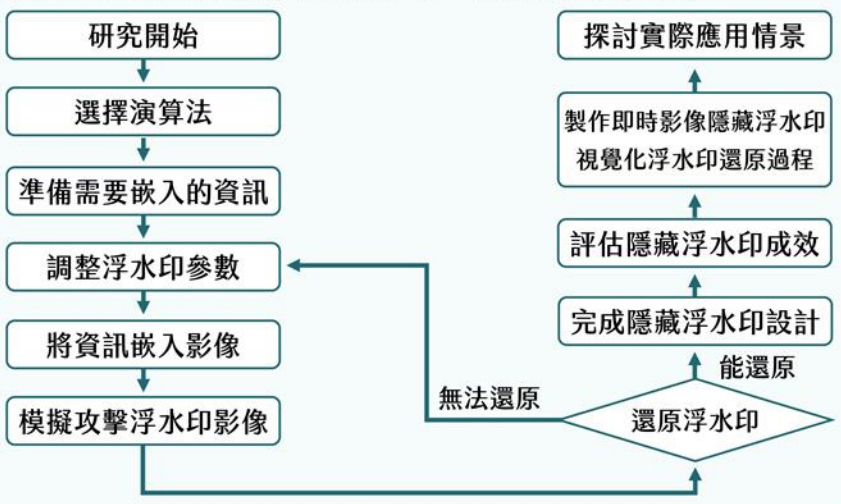
名稱	版本	描述
Windows 11	25375.1	作業系統。
Python	3.9.13	人工智慧、資料分析等領域常用的高階程式語言。
Pytorch	1.13.1+cu117	用於深度學習的函式庫，更加快速且易用。
Torchvision	0.14.1+cu117	PyTorch 配套函式庫，提供易用的影像處理工具。
OpenCV	4.7.0	電腦視覺的函式庫，提供多種影像處理的工具。
Windows 11 SDK	10.0.22000	微軟公司提供給 Windows 應用程式開發者的軟體開發包。
Visual Studio	17.5.1	微軟公司提供的整合開發環境（IDE）。
MSVC	14.35.32215	微軟公司提供的 C/C++ 編譯器及函式庫。
Davinci Resolve	18.1.4	影片剪輯軟體。

2 硬體設備

名稱	型號
中央處理器 (CPU)	AMD Ryzen 7 5800HS
記憶體 (RAM)	DDR4 3200MT/s 48GB
圖形處理器 (GPU)	GeForce RTX 3060 Laptop 6GB

研究流程及架構

先用 Python 撰寫程式碼，測試靜態影像演算法是否能有效抵禦動態影像會受到的攻擊，並評估調整各項參數對結果的影響。找出最佳參數組合之後，用 C++ 撰寫一套範例程式，測試靜態影像演算法用於動態影像上的執行速度及成效。



隱藏浮水印原理及測試方法

1 準備需要嵌入的資訊

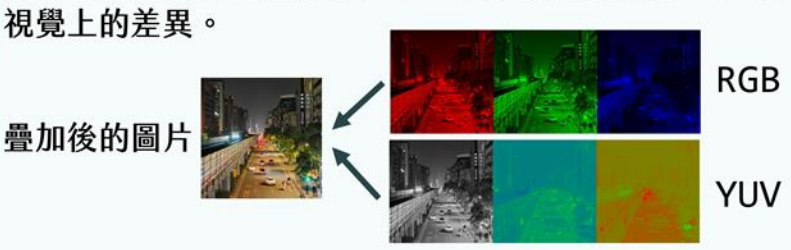
將嵌入的資訊轉成二進位序列形式。舉例：

```
T  W → 01010100 01010111
04 17 → 00000100 00010001
```

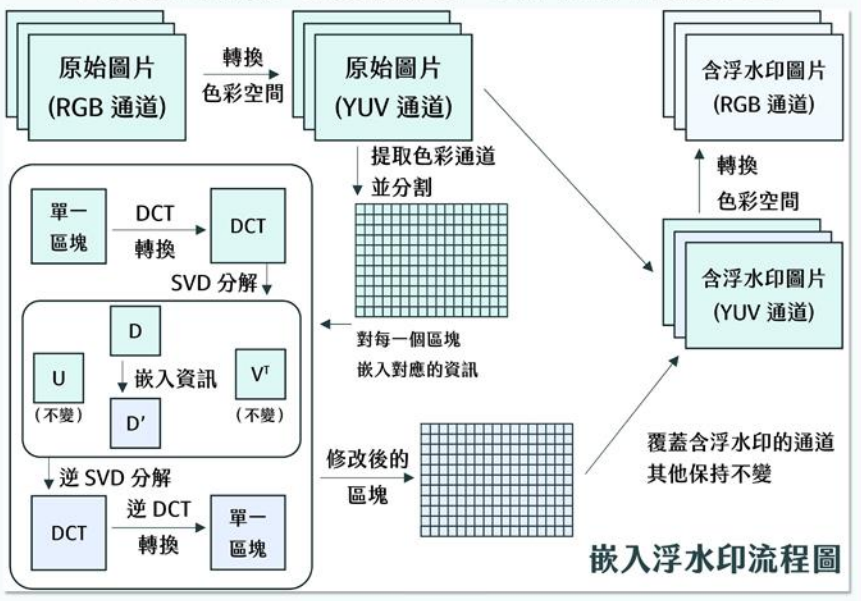
本研究隨機產生 22 個範圍 [0, 255] 的數字，總共 176 位元，做為測試資料。

2 將資訊嵌入影像

首先將目標圖片的色彩表示法由 RGB 轉換成 YUV 。RGB 以光的三原色：紅、綠、藍，表示某個顏色；YUV 則以 Y 通道表示明亮度，U、V 通道表示色度。因人眼對亮度較敏感，只對色度通道加上隱藏浮水印，不修改亮度通道，即可最小化視覺上的差異。



因為接下來的步驟比較繁瑣，以下為簡略的流程圖



提取轉換後的色度通道，分割成正方形區塊（Block）。

對每一個區塊，套用離散餘弦轉換（Discrete Cosine Transform, DCT），使「空間域」與「頻率域」能相互轉換。

轉換到頻率域能將資訊嵌入到肉眼不易察覺的特徵當中，並保持抗攻擊性。人眼對中低頻的資訊較為敏感，常見的圖片格式 JPEG ，經過轉換後，捨去高頻資訊以降低文件大小。在中低頻添加資訊，可有效的抵抗相同類型的影像壓縮。

對套用 DCT 後的區塊套用奇異值分解（Singular Value Decomposition, SVD），一種常用於矩陣的數值分析方法。

SVD 將一個矩陣 M 分解為三個矩陣 U, D, V^T，可分析影像的主要構成，圖片受到攻擊時，奇異值矩陣 D 中主要數值 不會大幅度的變動，以增加對於雜訊的容忍度。

過去隱藏浮水印是直接將轉換後長度為 l 的二進位序列 $B = \{b_0, b_1, \dots, b_{l-1}\},\ B \in 0, 1$ 照順序加入區塊中。定義品質係數 Q_1, Q_2，修改區塊編號 i 的第一、二個奇異值 $\gamma_{1,1}, \gamma_{2,2}$ ：

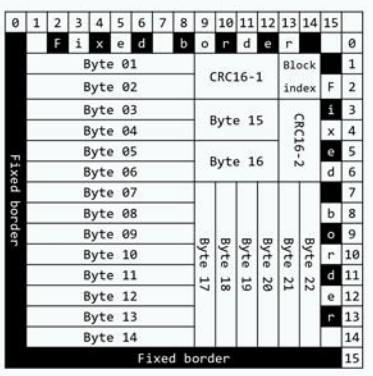
$$\gamma_{k,k} = Q_k\left[\left\lfloor \gamma_{k,k} \div Q_k \right\rfloor + 0.25 + 0.5 * B_{(i \bmod l)}\right], k \in 1, 2$$

若圖片遭到裁切，且不知道原始圖片的尺寸及位置，就沒辦法知道還原出來的區塊是代表哪一個區塊。舉例說明：

若還原時輸入原圖大小（左圖），可以從第一個位元（左上角）開始還原資訊。

若還原時輸入一塊截圖（右圖），就無從得知目前還原出來的區塊，在原圖中代表哪一部分的資訊。

因此本研究設計簡單、適合用於浮水印的二維 16x16 條碼。



Fixed border: 定位用的邊框。

Byte XX: 存放資料的區域，共可存放 22 位元組。

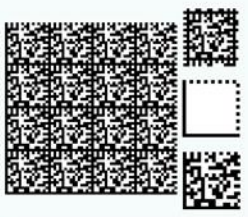
Block index: 標註條碼的編號，犧牲抗攻擊性，增加條碼數量以換取更大的資訊容量。

CRC16: 常用的循環冗餘校驗演算法，在此用於檢驗條碼正確性。

基於自行設計的條碼 M，條碼每個位元對每個區塊嵌入。修改整張圖片的 Block(i, j) 的第一、二個奇異值 $\gamma_{1,1}, \gamma_{2,2}$ ：

$$\gamma_{k,k} = Q_k\left[\left\lfloor \gamma_{k,k} \div Q_k \right\rfloor + 0.25 + 0.5 * M_{(i \bmod 16),(j \bmod 16)}\right], k \in 1, 2$$

將條碼從左上到右下放置。僅一個條碼的成果如下：



輸入原圖大小還原（左圖），條碼被整齊的嵌入。

當還原時輸入一塊截圖（右上），可從固定邊框（右中）定位條碼起始位置，找出從哪裡開始讀取資訊，定位後如右下。

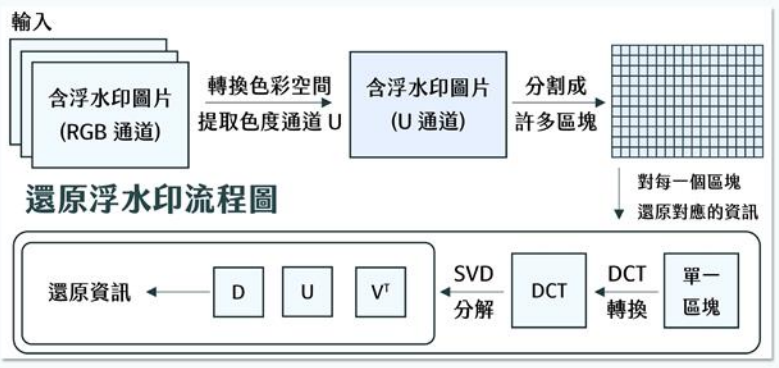
完成嵌入後，將所有區塊進行逆 SVD、逆 DCT，再覆蓋回原本的色度通道。YUV 轉換回 RGB ，即為含浮水印圖片。

3 生成模擬被攻擊的影像

測試圖片為電腦視覺領域常用的 Lenna ，以及 8 張手機拍攝的圖片。

不論是圖片或影片，在傳輸時經常受到不同程度的壓縮，或是遭到第三者修改。不同的攻擊手段都可能導致嵌入的浮水印，無法解碼正確結果。對每一張測試圖片，產生 12 張圖片，其中 1 張不修改及 11 種模擬影像遭修改的圖片，以測試影像遭到攻擊時的情況：

4 還原浮水印



還原浮水印流程圖

與嵌入浮水印類似，轉換成 YUV 色彩表示法，提取色彩通道，分割成區塊，對每一個區塊套用 DCT ，接著套用 SVD，得到 U, D, V^T 三個矩陣。品質係數 Q_1, Q_2 與嵌入資料時相同。

過去的隱藏浮水印還原時，須設定兩個變數，分別為與嵌入的資料相同長度 l 的二進位序列 $B = \{b_0, b_1, \dots, b_{l-1}\}$ 及紀錄區塊數量的陣列 $C = \{c_0, c_1, \dots, c_{l-1}\}$ 。讀取區塊編號 i 的奇異值 $\gamma_{1,1}$,$\gamma_{2,2}$ ：

$$\begin{cases} 若 d_1 \bmod Q_1 < {Q_1}/{2}, 令 A_1 = 0, 否則 A_1 = 1 \\ 若 d_2 \bmod Q_2 < {Q_2}/{2}, 令 A_2 = 0, 否則 A_2 = 1 \end{cases}$$

$$b_{(i \bmod l)} = b_{(i \bmod l)} + 0.75 \times A_1 + 0.25 \times A_2, \qquad c_i = c_i + 1$$

所有區塊完成後 ${b_i}/{c_i} < 0.5, 令 b_i = 0, 否則 b_i = 1$

本研究設計的條碼還原時，輸入長寬 H × W 的目標圖片，定義從每一個分割的區塊，還原的資料為一個長寬 ([H / (Block size)])×([W / (Block size)]) 的矩陣 M。讀取 Block(i, j) 的 D 矩陣的奇異值 $\gamma_{1,1}$,$\gamma_{2,2}$ ：

$$\begin{cases} 若 d_1 \bmod Q_1 < {Q_1}/{2}, 令 A_1 = 0, 否則 A_1 = 1 \\ 若 d_2 \bmod Q_2 < {Q_2}/{2}, 令 A_2 = 0, 否則 A_2 = 1 \end{cases}$$

$$M_{i,j} = 0.75 \times A_1 + 0.25 \times A_2$$

對所有 Block 進行完上述過程後，透過條碼邊框定位條碼。讀取每一個條碼的資料，計算 CRC16 並與讀取的 CRC16 比較，若兩者不同，則該條碼還原的資料是錯誤的。如果當時嵌入了超過一個條碼，還需讀取 Block index 以確認該條碼是第幾個條碼。

若每個條碼的資料皆錯誤，將 Block index 相同的條碼相加，為新條碼 N。若有 n 個 Block index 相同的條碼，操作使條碼只包含 0 / 1：

$$若 {M_{i,j}}/{n} < 0.5, 令 N_{i,j} = 0, 否則 N_{i,j} = 1$$

讀取條碼 N 的 data 1 ~ data 22，計算 CRC16，讀取條碼 N 的 CRC16，如果兩者不相同，則判定該影像無法正確還原任何資料。

5 評估嵌入浮水印品質

（一）浮水印還原能力 – 正確率 及 可還原資料圖片數

讀取每個區塊，比對該區塊嵌入時的資料，若相同則正確區塊數目加一。定義 **正確率 = 正確區塊數目 / 總區塊數目**，範圍為 [0, 1]。

「可還原資料圖片數」則是指整張圖片是否能還原出正確資料，越高的正確率及可還原資料圖片數，代表浮水印的抗攻擊能力越好。

（二）運算速度

在圖片讀取完成之後開始計時，直到嵌入浮水印完成，可以輸出圖片時，停止計時。單位為毫秒（ms）。

（三）峰值信噪比（Peak Signal to Noise Ratio, PSNR）

PSNR 單位為分貝（dB），數值越高，兩張圖片顏色越相似。

（四）結構相似度指標（Structural Similarity Index, SSIM）

SSIM 範圍為 [0, 1]，數值越大，圖片結構越相似，而當 SSIM = 1 時，兩張圖片相同。

隱藏浮水印成效

1 嵌入資料方式對比

評估自行設計的條碼除了裁切，是否在其它攻擊方式中，也具優勢。

因為有 CRC16 檢驗，可只提取單一有效條碼，故正確率相同時，可還原的圖片數較高。

嵌入方式	直接嵌入資料		嵌入條碼	
攻擊類型	準確率 / 可還原資料圖片數 ↑			
未攻擊	1.00	9	1.00	9
裁切	0.78	9	0.81	9
縮放	0.95	9	0.96	9
旋轉	0.91	9	0.92	9
亮度改變	0.58	2	0.52	3
色彩改變	0.68	4	0.67	6
高斯模糊	0.90	3	0.91	5
JPEG壓縮	0.96	9	0.97	9
覆蓋	0.88	9	0.89	9
椒鹽雜訊	0.93	9	0.89	9
H.264壓縮	0.77	0	0.85	3
AV1壓縮	0.81	2	0.88	4

2 YUV 色彩空間是否具有優勢、應使用哪個色彩通道

對單一色度通道嵌入浮水印，還原能力較好、運算快、影像品質更高。

手機圖片-2 局部放大圖

通道	RGB		YUV		Y		U		UV	
攻擊類型	準確率 / 可還原資料圖片數 ↑									
未攻擊	0.99	9	1.00	9	1.00	9	1.00	9	1.00	9
裁切	0.79	1	0.80	9	0.83	9	0.81	9	0.80	9
縮放	0.69	5	0.81	5	0.73	3	0.96	9	0.95	9
旋轉	0.80	9	0.86	9	0.81	3	0.92	9	0.91	9
亮度改變	0.45	0	0.50	0	0.50	0	0.52	3	0.55	2
色彩改變	0.57	0	0.56	1	0.64	1	0.67	6	0.60	3
高斯模糊	0.66	1	0.80	5	0.66	0	0.91	5	0.89	3
JPEG壓縮	0.85	5	0.96	9	1.00	8	0.97	9	0.95	7
覆蓋	0.87	9	0.89	9	0.89	8	0.89	9	0.88	9
椒鹽雜訊	0.54	0	0.67	0	0.52	0	0.89	9	0.93	9
H.264壓縮	0.43	0	0.53	0	0.33	0	0.85	3	0.77	1
AV1壓縮	0.45	0	0.64	0	0.51	0	0.88	4	0.81	2
耗時(ms) ↓	242.8		238.7		79.6		79.5		165.0	
含浮水印的未攻擊影像與原始影像的比較										
SSIM ↑	0.94		0.90		0.90		0.92		0.93	
PSNR (dB) ↑	30.79		30.45		35.81		35.23		34.23	

3 區塊大小對還原浮水印、影像品質、速度的影響

區塊大小 8 在抗攻擊能力、影像品質以及運算速度之間取得平衡。

手機圖片-2 局部放大圖

區塊大小	2		4		8		16	
攻擊類型	準確率 / 可還原資料圖片數 ↑							
原圖	1.00	9	1.00	9	1.00	9	1.00	9
裁切	0.81	9	0.81	9	0.81	9	0.76	0
縮放	0.82	0	0.95	8	0.96	9	0.84	6
旋轉	0.85	0	0.93	9	0.92	9	0.86	6
亮度改變	0.87	9	0.71	6	0.52	3	0.52	0
色彩改變	0.84	9	0.74	6	0.67	6	0.59	0
高斯模糊	0.71	0	0.79	0	0.91	5	0.76	0
JPEG壓縮	0.83	0	0.90	0	0.97	9	0.83	0
覆蓋	0.89	9	0.89	8	0.89	9	0.87	0
椒鹽雜訊	0.96	9	0.97	9	0.89	9	0.74	3
H.264壓縮	0.75	0	0.80	1	0.85	3	0.71	0
AV1壓縮	0.80	0	0.85	1	0.88	4	0.66	0
耗時(ms) ↓	983.1		265.2		79.5		48.7	
含浮水印的未攻擊影像與原始影像的比較								
SSIM ↑	0.77		0.89		0.92		0.99	
PSNR (dB) ↑	25.45		31.63		35.23		41.60	

4 品質係數對還原浮水印、影像品質、速度的影響

若要能抵抗影片編碼攻擊需要 $Q_1, Q_2 = 60$，但影像品質會略為下降。

手機圖片-2 局部放大圖

Q_1, Q_2	20, 20		30, 30		40, 40		50, 50		60, 60	
攻擊類型	準確率 / 可還原資料圖片數 ↑									
原圖	1.00	9	1.00	9	1.00	9	1.00	9	1.00	9
裁切	0.80	0	0.81	9	0.76	9	0.81	9	0.81	9
縮放	0.78	0	0.84	7	0.92	9	0.96	9	0.98	9
旋轉	0.84	0	0.88	9	0.90	9	0.92	9	0.92	9
亮度改變	0.53	0	0.51	0	0.54	1	0.52	3	0.55	3
色彩改變	0.58	0	0.63	2	0.69	2	0.67	6	0.74	7
高斯模糊	0.67	0	0.73	0	0.85	0	0.91	5	0.93	5
JPEG壓縮	0.71	0	0.81	0	0.90	4	0.97	9	0.99	9
覆蓋	0.89	9	0.89	9	0.86	9	0.89	9	0.89	9
椒鹽雜訊	0.74	1	0.82	4	0.89	7	0.89	9	0.92	9
H.264壓縮	0.62	0	0.64	0	0.72	2	0.85	3	0.90	4
AV1壓縮	0.59	0	0.63	0	0.69	1	0.88	4	0.92	6
含浮水印的未攻擊影像與原始影像的比較										
SSIM ↑	0.99		0.98		0.97		0.92		0.94	
PSNR (dB) ↑	43.30		40.78		38.76		35.23		36.9	

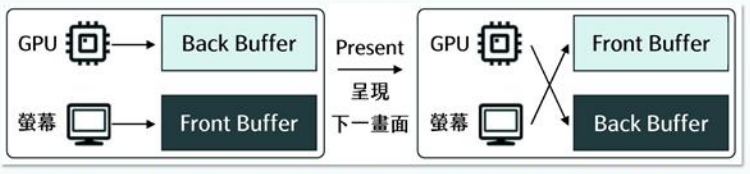
即時影像隱藏浮水印

1 原理

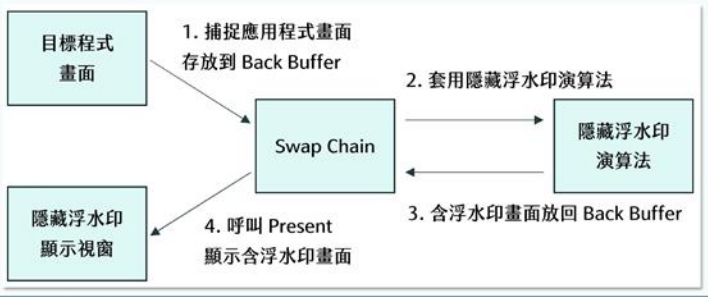
將演算法套用到一個範例上，以驗證該演算法對於即時動態影像添加浮水印的可行性。本研究在電腦上撰寫程式，捕捉特定視窗的畫面，並即時輸出含有隱藏浮水印的畫面。為了方便以人眼對比嵌入浮水印前後的畫面，本程式新建一個視窗顯示含隱藏浮水印的畫面。

用 C++ 撰寫程式捕捉畫面，呼叫 Windows.Graphics.Capture API 使程式能夠捕捉特定視窗的畫面，並且存放到 Direct3D 11 Swap Chain（交換鍊）的 Back Buffer（後緩衝區）中。Direct3D 11 是一個用於 Windows 平台的圖形 API，其中 Swap Chain 是與影像顯示高度相關的一個概念。

Swap Chain 中通常具有多個 Buffer，Front Buffer（前緩衝區）是螢幕正在顯示的內容，而 Back Buffer 則是 GPU（圖形處理器）完成渲染（Render）的畫面。當新的一個畫面 Present（呈現）時，Swap Chain 交換兩 Buffer 的指標，兩者「功能」互換，但「內容（存放的畫面）」不變。



讀取 Back Buffer 中的內容，套用隱藏浮水印演算法，再寫入到 Back Buffer 中，最後呼叫 Present，讓本程式顯示已經加上浮水印的畫面，即完成即時加入浮水印的過程。



2 評估方式

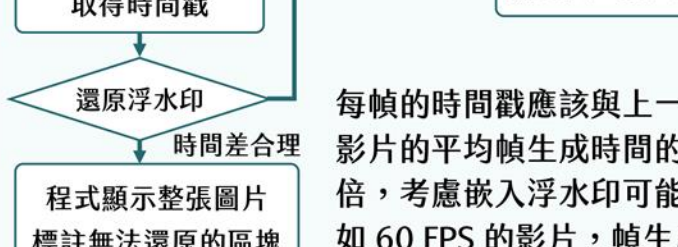
設定程式擷取影片播放軟體 PotPlayer，播放一段以解析度 2560x1440，120 FPS 錄製的遊戲影片。測量程式每幀畫面輸出平均間隔（幀生成時間，單位：毫秒 ms），以及嵌入浮水印消耗的平均時間，以評估程式執行速度，找出影響速度的主要因素。每秒幀數（Frames Per Second, FPS）= (1000 / 幀生成時間)。電影為 24 FPS，電腦螢幕為 60 FPS。幀生成時間越低，FPS 越高越好，運算速度越快、畫面越流暢。

3 成效

首先測量即時影像隱藏浮水印程式的輸出，與前述模擬攻擊、評估參數時靜態圖片的輸出是否不同，結果為 SSIM = 1 且 PSNR = 80，代表輸出相同，前述實驗測試的演算法具有抗攻擊能力，在即時影像隱藏浮水印也適用。

不嵌入浮水印，幀生成時間（Frame Time）仍大致相同，說明程式執行瓶頸為畫面捕捉，而不是嵌入浮水印。僅考慮對影像嵌入浮水印，可以達到約 450 FPS ，遠高於一般使用者 60 FPS 的需求，說明即時對影像嵌入浮水印效能上是可行的。

變因	平均幀生成時間 ↓	平均浮水印嵌入時間 ↓	平均 FPS ↑
不嵌入	18.22 ms	0 ms	54.88
嵌入浮水印	18.38 ms	2.17 ms	54.41



左方為被捕捉畫面的視窗，右方為含浮水印的畫面。

紅色方框中為幀生成時間。數據不同是因截圖時需縮小視窗，解析度不同導致。

浮水印還原結果視覺化

1 原理

除了從影像中提取資訊外，此演算法也可以應用於偵測畫面是否遭到人為修改，在實際生活中，可應用於法院偵查，以證明提供的影片是原始檔案，未經任何修改，是有效的證物。

在嵌入浮水印的過程中，儲存嵌入時的 Timestamp（時間戳），用條碼可嵌入的 22 位元組資訊，其中的 6 位元組，以 4 位元組儲存整數部分（秒），以 2 位元組儲存小數部分（毫秒），剩下 16 位元組可依使用者需求儲存其他資訊。製作程式，還原影片每個畫面的浮水印，並進行以下判斷：

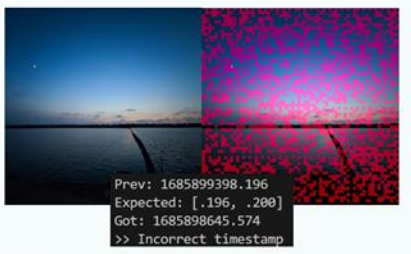
每幀的時間戳應該與上一幀差距約在影片的平均幀生成時間的 0.5 ~ 2.5 倍，考慮嵌入浮水印可能延遲。
如 60 FPS 的影片，幀生成約為 16.6 毫秒，即兩幀畫面之間，時間戳的差應該約在 8 ~ 41 毫秒之間。

2 實際成果

為節省空間將影片的解析度降低
沒有修改其他部分
程式無標註錯誤區塊

影片編碼並非人為修改
因演算法無法完全抵抗影片壓縮
部分區塊被錯誤的標註

上半部分透過調色
來掩蓋原本圖片的細節
程式標註錯誤區塊



不同時間嵌入但可還原浮水印的兩張圖片，程式偵測時間戳無效

⑧ 結論與實際應用

1 研究成果

現有的隱藏浮水印大多針對靜態影像，尚未有將隱藏浮水印應用在動態影像上的案例，也未測試動態影像編碼攻擊。

本研究成功發展出基於 DCT － SVD 演算法，針對動態影像的即時隱藏浮水印系統。透過找出最佳使用參數、模擬影像被攻擊的情況，評估浮水印還原能力、影像品質、運算速度，證明此演算法在影像受到不同干擾時，能正確還原資訊。

2 改進還原能力及增強資料安全性

透過修改資訊嵌入的方式，設計簡易的條碼，解決過去隱藏浮水印需要預先知道原始圖像大小，無法直接還原的問題，增加隱藏浮水印還原能力。可自行設計條碼定義，未知該定義的第三者無法透過條碼還原出有效的資訊，以保護資料安全。

3 實際應用：即時影像隱藏浮水印

實作的程式使用 Direct3D 11 API 捕捉及返回畫面，且執行速度能滿足需求，可輕易整合進各種會用到該 API 的程式，如影音播放軟體、遊戲等，具更多、更廣的應用價值，可保護具有著作權與智慧財產權的軟體、遊戲、影片等。

4 實際應用：隱藏浮水印還原結果視覺化

透過偵測隱藏浮水印的破壞程度、檢查隱藏浮水印嵌入的時間，以畫面形式告知使用者畫面是否有被人為竄改或替換。此技術可用於監視錄影，以證明畫面未曾遭受惡意修改。

品 未來展望

1 尋找更穩健的演算法

面對影片編碼壓縮攻擊，有時無法還原浮水印，或必須犧牲畫面品質。希望未來能結合其他頻率域－空間域轉換方式，尋找適用於動態影像，具抗攻擊能力與影像品質高的演算法。

2 運用糾錯演算法增強抗攻擊能力

影片編碼壓縮會破壞條碼特定區塊，導致整個條碼無效。希望未來能結合 Reed-Solomon 糾錯或其他糾錯演算法，使部分區域被破壞時，也能還原出正確資訊，增強抗攻擊能力。

冂 參考文獻資料

[1] KTinker, "《暗黑破壞神 4》Alpha 內部測試版遊戲畫面外流，曝光創角與難度選項," 9 8 2022. [Online]. Available: https://today.line.me/tw/v2/amp/article/OpZeJe6. [Accessed 19 3 2023].

[2] Rocky, "《俠盜獵車手 6》高達 90 多部早期開發影片遭駭客大量洩漏，這應該是遊戲歷史上最大洩密事件之一," 19 9 2022. [Online]. Available: https://www.kocpc.com.tw/archives/460111. [Accessed 19 3 2023].

[3] KKVincent, "Take-Two 執行長稱《俠盜獵車手 6》外流事件對商業無影響，但對開發者造成了「情緒傷害」," 11 2 2023. [Online]. Available: https://www.kocpc.com.tw/archives/480036. [Accessed 19 3 2023].

[4] [deleted], "r/xboxone - One way MS tracked NDA breakers in the 360 era," 10 12 2018. [Online]. Available: https://old.reddit.com/r/xboxone/comments/a4y3n8/one_way_ms_tracked_nda_breakers_in_the_360_era/. [Accessed 14 3 2023].

[5] Shweta Wadhera, Deepa Kamra, Ankit Rajpal, Aruna Jain, Vishal Jain, "A Comprehensive Review on Digital Image Watermarking," in WCNC-2021: Workshop on Computer Networks & Communications, 2021.

[6] P. B. Dasgupta, "Algorithmic Analysis of Invisible Video Watermarking using LSB Encoding over a Client-Server Framework," International Journal of Computer Trends and Technology (IJCTT), vol. 36, no. 3, 2016.

[7] 辜慶軒, 胡懷祖, 徐鈴淵, "於 DWT-SVD-DCT 轉換域所發展之影像浮水印技術," International Journal of Advanced Information Technologies(IJAIT), vol. 8, no. 1, 2014.

[8] N. Ahmed, T. Natarajan, K. R. Rao, "Discrete Cosine Transform," IEEE Transactions on Computers, vol. 100(1), pp. 90-93, 1974.

[9] Chin-Chen Chang, Piyu Tsai, Chia-Chen Lin,, "SVD-based digital image watermarking scheme," Pattern Recognition Letters, vol. 26, no. 10, pp. 1577-1586, 2005.

[10] Begum, Mahbuba, Mohammad Shorif Uddin, "Analysis of digital image watermarking techniques through hybrid methods," Advances in Multimedia, vol. 2020, pp. 1-12, 2020.

[11] Rahman, Md. Maklachur, "A DWT, DCT and SVD Based Watermarking Technique to Protect the Image Piracy," International Journal of Managing Public Sector Information and Communication Technologies, vol. 4, 2013.

[12] "RGB-YUV Conversion," [Online]. Available: https://www.cs.sfu.ca/mmbook/programming_assignments/additional_notes/rgb_yuv_note/RGB-YUV.pdf. [Accessed 19 3 2023].

[13] @guofei9987, "GitHub - guofei9987/blind_watermark," 3 12 2022. [Online]. Available: https://github.com/guofei9987/blind_watermark/tree/839af4c7850db1921108d30cf4dc92c8e8ebb130. [Accessed 27 2 2022].

[14] @lishuaijuly, "GitHub - lishuaijuly/watermark," 5 11 2017. [Online]. Available: https://github.com/lishuaijuly/watermark/tree/23d167192aa90dcb329348174211eead1367d38. [Accessed 27 2 2023].

[15] @ShieldMnt, "GitHub - ShieldMnt/invisible-watermark," 16 11 2022. [Online]. Available: https://github.com/ShieldMnt/invisible-watermark/tree/0690c09a9ec6e665b5072aa11deea12467e47afe. [Accessed 27 2 2023].

[16] "The Rest of the Lenna Story," 3 11 2001. [Online]. Available: http://www.lenna.org/. [Accessed 28 2 2023].

[17] Y. Liu, "AV1 beats x264 and libvpx-vp9 in practical use case," 10 4 2018. [Online]. Available: https://engineering.fb.com/2018/04/10/video-engineering/av1-beats-x264-and-libvpx-vp9-in-practical-use-case/. [Accessed 19 3 2023].

[18] Alain Horé, Djemel Ziou, "Image quality metrics: PSNR vs. SSIM," in ICPR 2010, 2010.

[19] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, Eero P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, APRIL 2004.

[20] @xinntao, "BasicSR/basicsr/metrics · XPixelGroup/BasicSR," 8 9 2022. [Online]. Available: https://github.com/XPixelGroup/BasicSR/tree/b0ee3c8/basicsr/metrics#psnr-%E5%92%8C-ssim. [Accessed 27 2 2023].

[21] I. The MathWorks, "Structural similarity (SSIM) index for measuring image quality," 2021. [Online]. Available: https://www.mathworks.com/help/images/ref/ssim.html. [Accessed 27 2 2022].

[22] @ajbennet, @robmikh, "Windows.UI.Composition-Win32-Samples/cpp/ScreenCaptureforHWND," 10 1 2023. [Online]. Available: https://github.com/Microsoft/Windows.UI.Composition-Win32-Samples/tree/830f854/cpp/ScreenCaptureforHWND. [Accessed 28 2 2023].