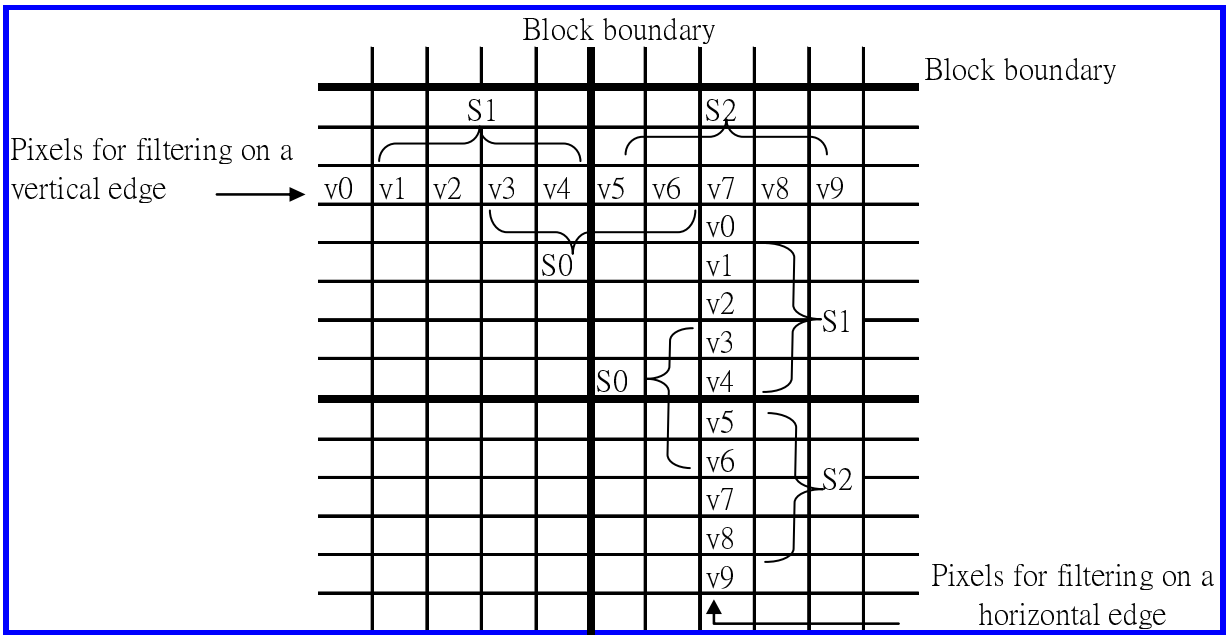


圖 27 :MPEG4 deblocking Method

上圖(圖27)是MPEG4去方塊效應演算法的資料判別圖,利用每個方塊左右5個像素的值來做平滑度的判斷,判斷方法如下:

$$eq\_cnt = \phi(v0-v1)+\phi(v1-v2)+\phi(v2-v3)+\phi(v3-v4)$$
$$+\phi(v4-v5)+\phi(v5-v6)+\phi(v6-v7)+\phi(v7-v8)+\phi(v8-v9),$$
$$where \quad \phi(\gamma)=1 \; if \; |\gamma| \le THR1 \text{ and } 0 \text{ otherwise.}$$

If $(eq\_cnt \ge THR2)$

DC offset mode is applied,

else

Default mode is applied.

endif

THR1=2 and THR2=6.

利用每個像素的差來做判斷,如果兩兩像素的差大於一定的閥值(THR1)的話,$\phi(\gamma)$就會等於1, 如果每個方塊左右5個像素超過THR1的數目大過THR2的數目的話,就代表這個方塊和相鄰的方塊相差太多,必須使用所謂DC offset mode

來做去方塊效應的處理,否則的話,就只做Default的去方塊效應處理,下面先說明什麼是Default的方塊效應處理.

Default mode

$v_4' = v_4 - d,$

$v_5' = v_5 - d,$

$d = CLIP(5 \cdot (a_{3,0}' - a_{3,0})//8, 0, (v_4 - v_5)/2) \cdot \delta(|a_{3,0}| < QP)$

$where\ a_{3,0}' = SIGN(a_{3,0}) \cdot MIN(|a_{3,0}|, |a_{3,1}|, |a_{3,2}|).$

$a_{3,0} = ([2,-5,5,-2] \bullet [v_3, v_4, v_5, v_6]^T)//8,$

$a_{3,1} = ([2,-5,5,-2] \bullet [v_1, v_2, v_3, v_4]^T)//8,$

$a_{3,2} = ([2,-5,5,-2] \bullet [v_5, v_6, v_7, v_8]^T)//8,$

$CLIP(x, p, q)$: *clips* x to a value between p and q,

QP:quantisation parameter of the macroblock where pixel v_5 belongs.

δ(condition)=1,if the "condition" is true and 0 otherwise.

// : rounding the the nearest integer.

由上述的程式可以知道,其實Default 的去方塊效應處理就是只有針對每個方塊的邊界點做處理(也就是v_4'和v_5'),希望處理完的v_4'和v_5'會呈現平滑的現象,所以要求出要減去的值(d),但d會跟其他像素的值有關係,在這是用非線性的處理,先算出3種預定位置的值($a_{3,0}$, $a_{3,1}$, $a_{3,2}$),再來做Clip的動作,其中要考慮的還有每個方塊所用的QP值,(也就是Quantization Parameter, $1 \le QP \le 31$),因為在我們的比較中,所用的影像皆是由JPEG而來,所以換成MPEG4的QP,其值為31.

而如果判斷這個方塊是十分平緩變化的區域的要,就要用DC offset mode來做。

以下是DC offset mode的程序:

DC offset mode:

$$\max = MAX(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8),$$

$$\min = MIN(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8),$$

$$if\ (|\max - \min\} < 2 \cdot QP)\{$$

$$v_n' = \sum_{k=-4}^{4} b_k \cdot p_{n+k}, 1 \le n \le 8,$$

$$p_m = \begin{cases} (|v_1 - v_0| < QP)\ ?\ v_0 : v_1, if\ \text{m<1} \\ v_m, if\ 1 \le \text{m} \le 8 \\ (|v_8 - v_9| < QP)\ ?\ v_9 : v_8, if\ \text{m>8} \end{cases}$$

$$\{b_k : -4 \le k \le 4\} = \{1,1,2,2,4,2,2,1,1\}//16$$

}

else

No change will be done

因為DC offset mode是對付比較平滑的區域,所以要改變的像素當然就多了,由 $v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8$ 皆要被重新計算,其也是一個非線性的方法搭配一個濾波器的係數來做.

接下來,我們便利用Matlab來實現這五種去方塊效應的演算法,並利用10張512x512的灰階圖像來做比較.

4.2. 比較結果

比較的文檔是用了10張圖,每張圖的排版方式是依順上左(a)為壓縮後的圖,上右為(b)POCS去方塊效應演算法的結果,中左為(c)二維小波去方塊效應演算法的結果, 中右為(d)一維小波[7]去方塊效應演算法的結果,下左(e)MPEG4去方塊效應演算法,下右(f)為本演算法的結果.

(a)

(b)

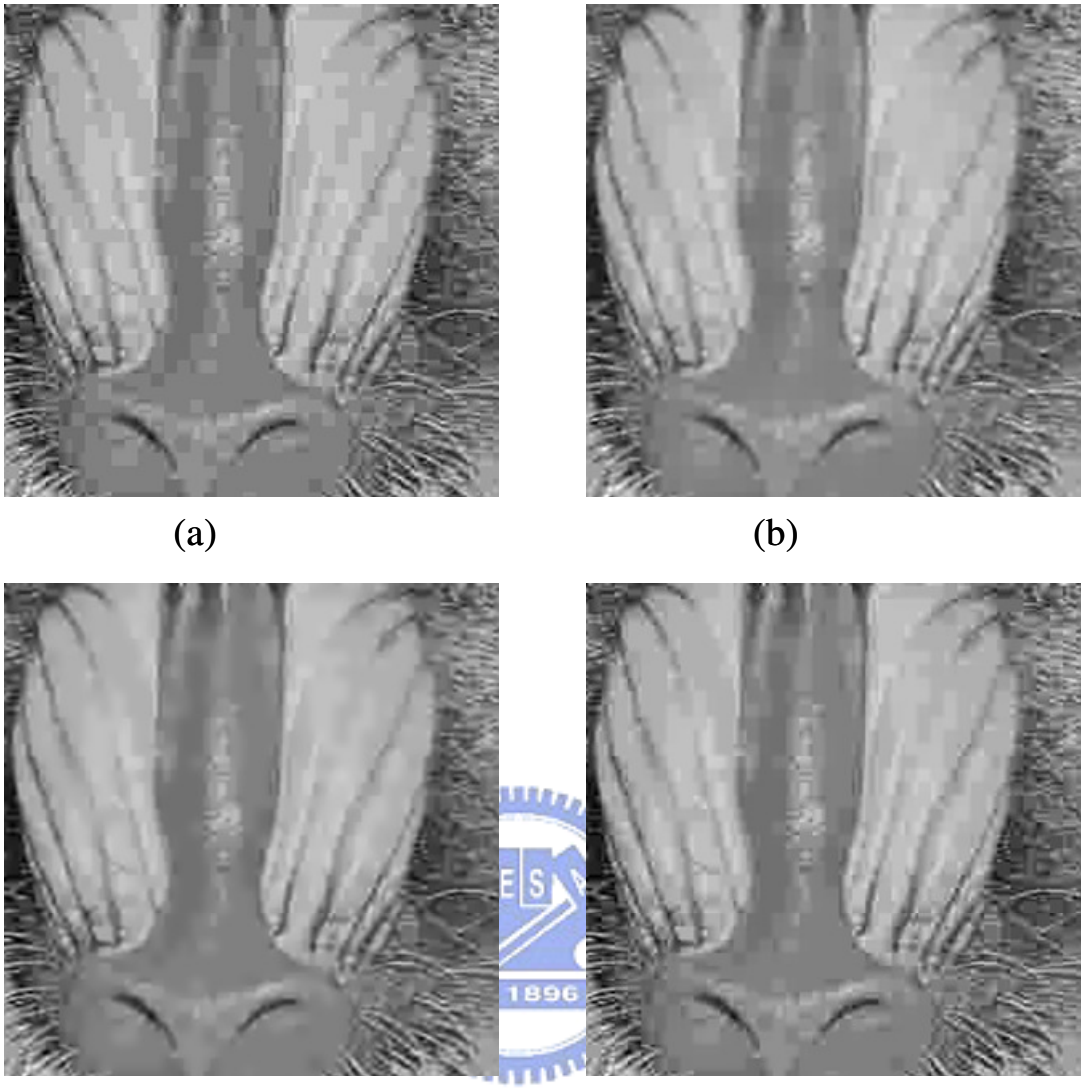
(c)

(d)

(e)

(f)

圖 28:各種去方塊演算法的比較(lena)
(a) PSNR=28.47 (b) PSNR=27.81
(c) PSNR=27.98 (d) PSNR=28.80
(e) PSNR=28.65 (f) PSNR=29.55



(a) (b)

(c) (d)

(e) (f)

圖 29:各種去方塊演算法的比較(baboon)
(a) PSNR=22.16 (b) PSNR=21.66
(c) PSNR=22.28 (d) PSNR=22.21
(e) PSNR=22.09 (f) PSNR=22.16

(a)

(b)

(c)

(d)

(e)

(f)

圖 30:各種去方塊演算法的比較(peppers)

(a) PSNR=28.18(b) PSNR=27.491

(c) PSNR=27.02(d) PSNR=28.56

(e) PSNR=28.46(f) PSNR=29.20

(a)

(b)

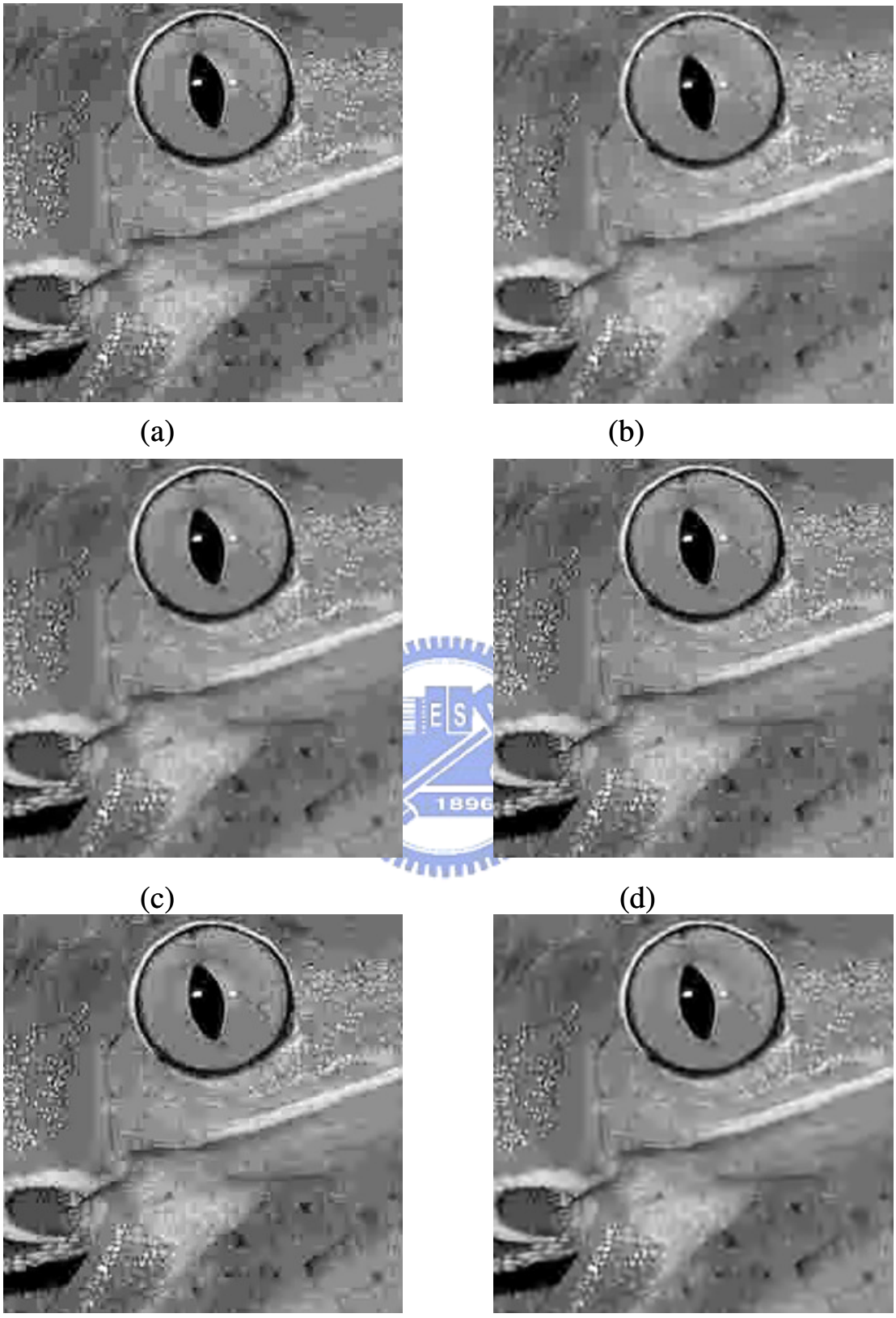
(c)

(d)

(e)

(f)

圖 31 :各種去方塊演算法的比較(boat)
(a) PSNR=27.05(b) PSNR=26.14
(c) PSNR=26.50(d) PSNR=27.27
(e) PSNR=27.12(f) PSNR=27.77

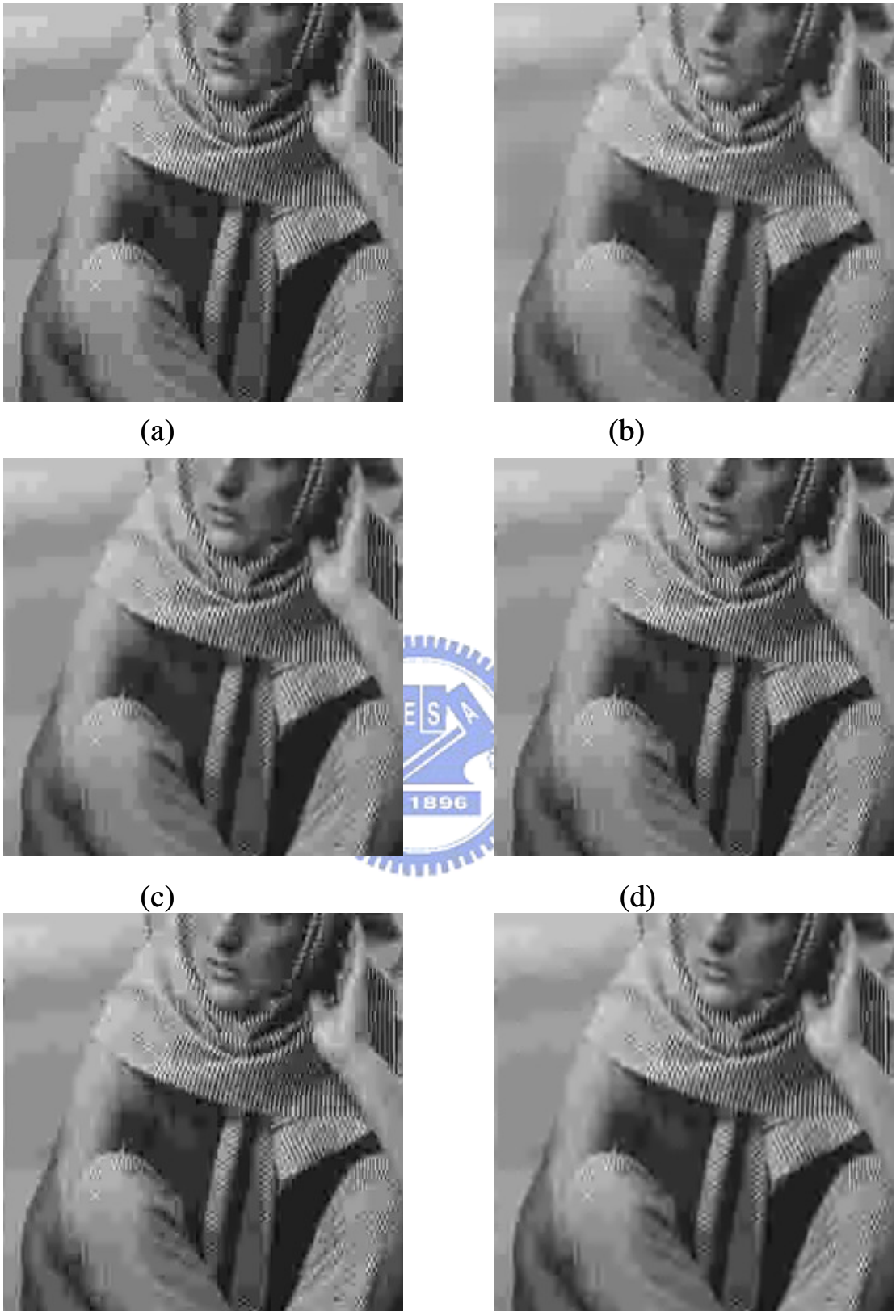


(a) (b)

(c) (d)

(e) (f)

圖 32:各種去方塊演算法的比較(frog)
(a) PSNR=23.66(b) PSNR=23.19
(c) PSNR=23.66(d) PSNR=23.76
(e) PSNR=23.66(f) PSNR=24.03

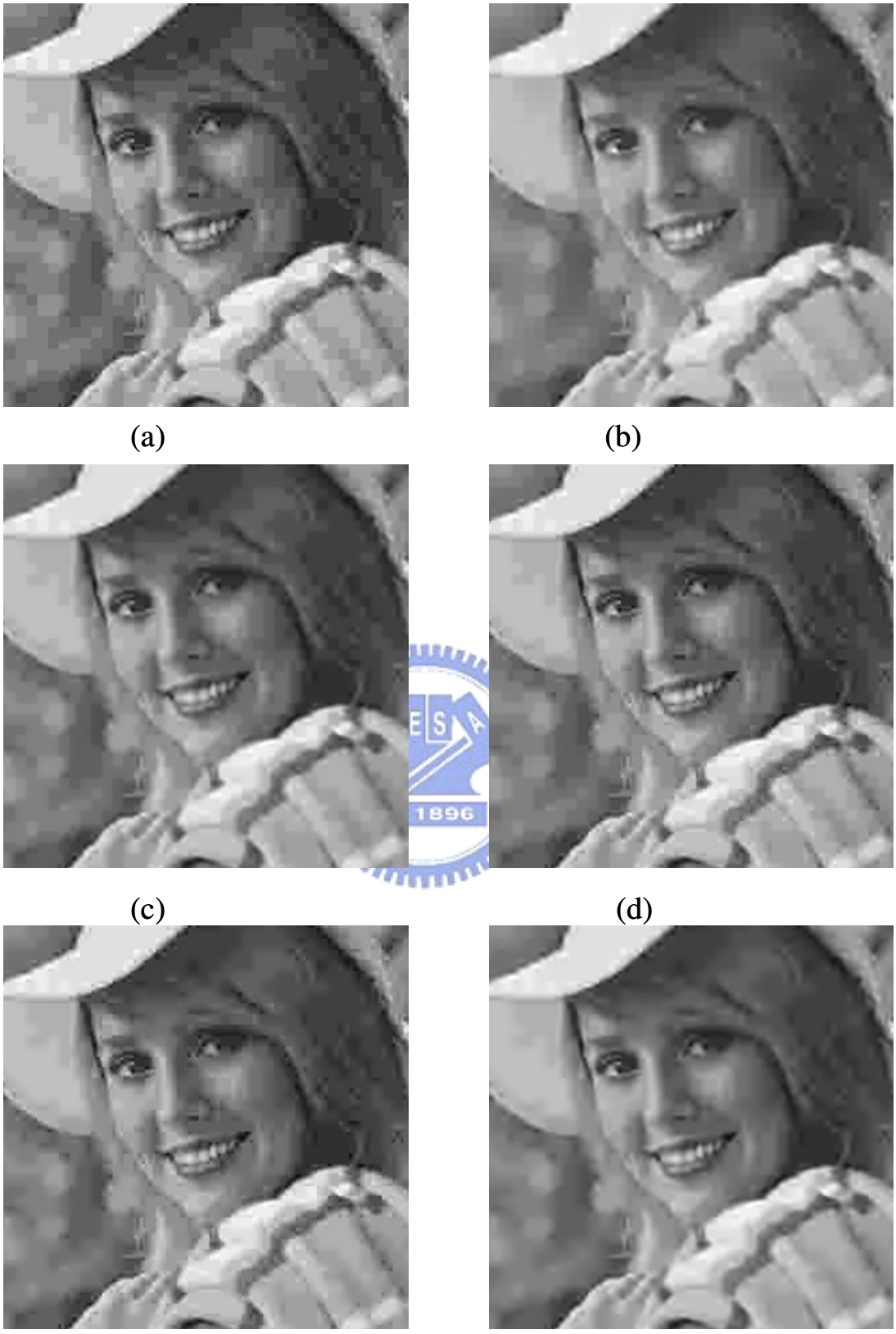


(a) (b)

(c) (d)

(e) (f)

圖 33:各種去方塊演算法的比較(barb)
(a) PSNR=24.18(b) PSNR=23.71
(c) PSNR=24.11(d) PSNR=24.32
(e)PSNR=24.24 (f) PSNR=24.49



(a) (b)

(c) (d)

(e) (f)

圖 34:各種去方塊演算法的比較(elain)
(a) PSNR=28.36(b) PSNR=27.74
(c) PSNR=27.70.(d) PSNR=28.85
(e) PSNR=28.81(f) PSNR=29.53

(a)

(b)

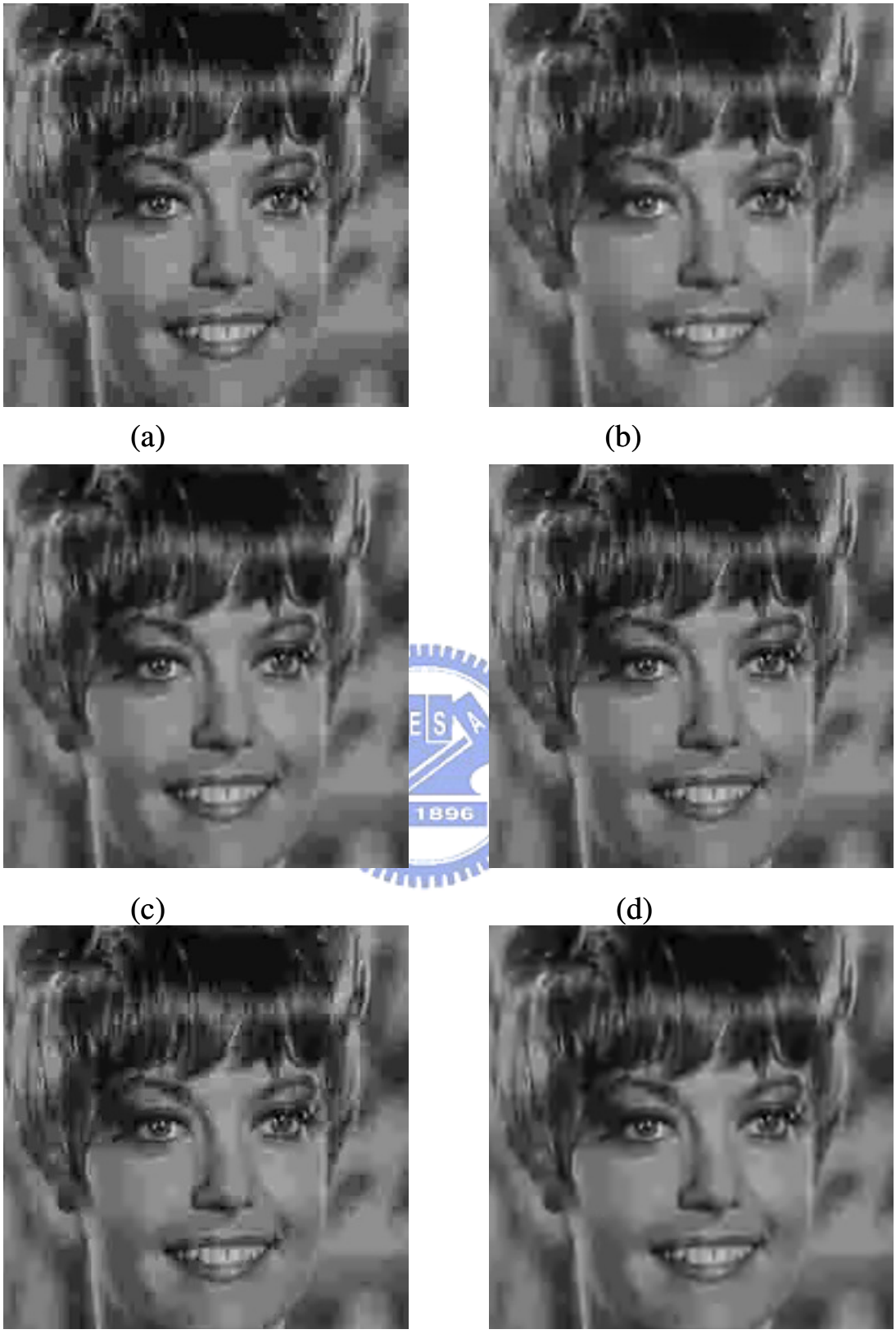
(c)

(d)

(e)

(f)

圖 35:各種去方塊演算法的比較(Tank)
(a) PSNR=27.99(b) PSNR=27.46
(c) PSNR=27.19.(d) PSNR=28.39
(e) PSNR=28.34 (f) PSNR=28.81

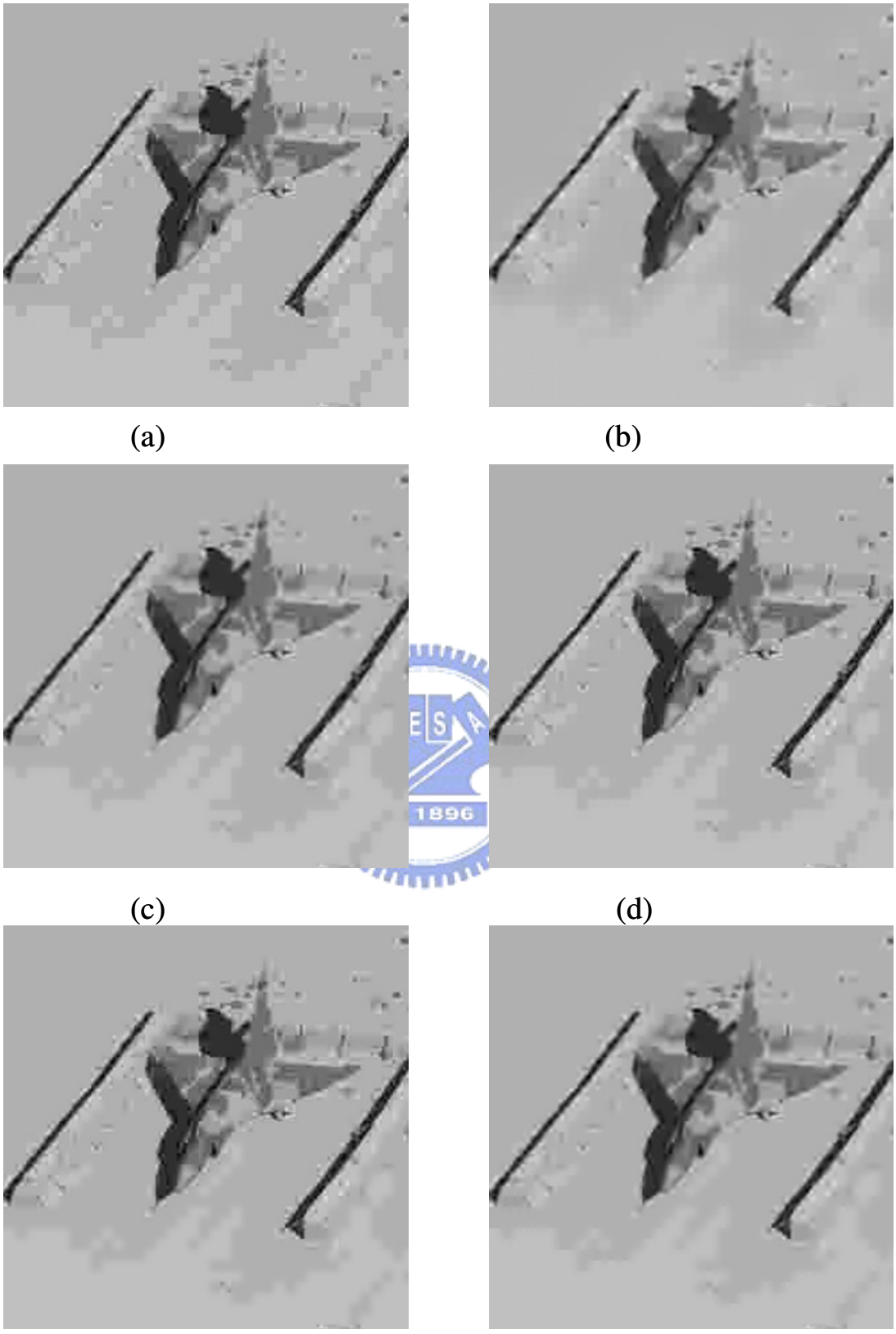


(a) (b)

(c) (d)

(e) (f)

圖 36:各種去方塊演算法的比較(zelda)
(a) PSNR=29.81(b) PSNR=29.15
(c) PSNR=30.01.(d) PSNR=30.51
(e) PSNR=30.50 (f) PSNR=31.41



(a) (b)

(c) (d)

(e) (f)

圖 37:各種去方塊演算法的比較(Airplane)
(a) PSNR=30.83 (b) PSNR=31.46
(c) PSNR=27.89.(d) PSNR=31.16
(e) PSNR=31.10 (f) PSNR=31.53

由圖28至圖37可以看出,本文的改進方法可以有效的去方塊效應,而且在PSNR的方面也有所改進,表3列出其PSNR的比較.

	original	POCS	2-D	1-D	MPEG4	ours
lena	28.47	27.81	27.98	28.8	28.65	29.55
baboon	22.16	21.66	22.28	22.21	22.09	22.16
peppers	28.18	27.491	27.02	28.56	28.46	29.2
boat	27.05	26.14	26.5	27.27	27.12	27.77
frog	23.66	23.19	23.66	23.76	23.66	24.03
barb	24.18	23.71	24.11	24.32	24.24	24.49
elain	28.36	27.74	27.7	28.85	28.81	29.53
Tank	27.99	27.46	27.19	28.39	28.34	28.81
zelda	29.81	29.15	30.01	30.51	30.5	31.41
Airplane	30.83	31.46	27.89	31.16	31.1	31.53
couple	25.99	25.06	26.05	26.21	26.12	26.23
goldhill	27.04	26.2	26.46	27.33	27.22	27.83
aerial	24.09	23.48	23.49	24.16	23.98	24.63

表格 3:各種去方塊效應的 PSNR 的比較

由表 3 可以看出在 PSNR 量化的表現上,本文所提出的演算法在對原本有方塊效應的影像有不錯的改進,除了 baboon 這個影像外,都會有 0.3~1.1db 的提升,而和其餘的演算法比起來,也是都是提升 PSNR 最多的(見表 3 灰色區域). 因為 baboon 本身是有很多邊緣的影像, 而所有演算法都有對影像本身做過很多處理,所以 PSNR 的算法在對很多邊緣的影像的方面可以看出來是無法正確反應出來的.這也是在量化標準和主觀標準上有很大的不同.

表格 4 則是對每一種演算法在 Matlab 上的執行時間來做一個比較.

	POCS(秒)	2-D(秒)	1-D(秒)	MPEG4(秒)	ours(秒)
lena	115.66	7.51	17.39	7.56	27.59
baboon	122.63	5.44	20.37	5.72	29.31
peppers	117.95	5.56	19.63	6.55	28.16
boat	128.38	5.3	18.63	6.17	28.74
frog	147.33	5.48	21.38	5.71	28.73
barb	123.63	5.3	18.61	6.06	28.58
elain	154.88	5.33	20.53	5.93	28.38
Tank	136.2	5.24	19.39	6.73	28.16
zelda	159.4	5.36	19.64	5.92	27.17
Airplane	128.18	5	18.86	6.37	29.23
couple	146.54	5.49	19.22	5.79	28.86
goldhill	161.83	5.49	20.16	5.8	28.33
aerial	138.24	5.56	19.99	5.47	29.16

表格 4:各演算法的 Matlab 執行時間

由表 4 可以看出,POCS 因為是遞迴的關係,所以執行時間最長,而在二維的小波[18]去方塊演算法中,因為 MATLAB 的運算事實上是以陣列(array) 及矩陣(matrix) 方式在做運算執行,所以執行時間上比一維的小波[7]去方塊演算法還快,但要注意的是,這不代表在硬體實踐上二維的會比一維的快且小.而注意的是在 MPEG4 的去方塊演算法在 Matlab 確實是有很短的執行時間,但因為 Matlab 在對某些運算如求最大值(max),最小值(min),等的運算是比較快的,所以並沒有完成反應出真實在硬體上設計的難度,最後要看的是本文的運算速度,是比原來[7]的慢了約 0.5 倍,但這是因為本文的 matlab 是有加入 donoho 的做法,所以造成的執行時間會有所延遲.綜合上所述,除了 POCS 是很長的計算時間外,其實的 4 種演算法的計畫時間都在可容忍的程度之內.

如果是用主觀的比較法,也就是利用人眼來看這五種去方塊效應的結果,

可以看出,雖然 POCS 是用遞迴的方式來做,其效果其不如本文提出的演算法的結果,本文提出來的演算法,不論是在影像內容複雜度或是平滑的影像,都可以有效的去除方塊效應,且不會太過模糊化重構的影像,和原來[7]的比較,也有很大的進步,和二維的做法[18]比起來,也可以看出二維的做法在重構完也會有沒有完成去方塊效應的地方,MPEG4 的去方塊的效果普遍來說不錯,但是其缺點就是並不是全部的影像的方塊效應都有解決,會有一些地方仍是會有很明顯的方塊效應存在,所以以主觀來看,本文是很有效的去除了方塊效應.