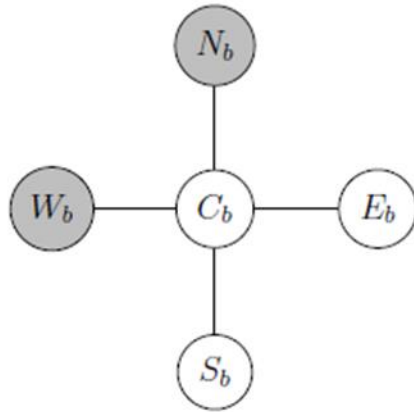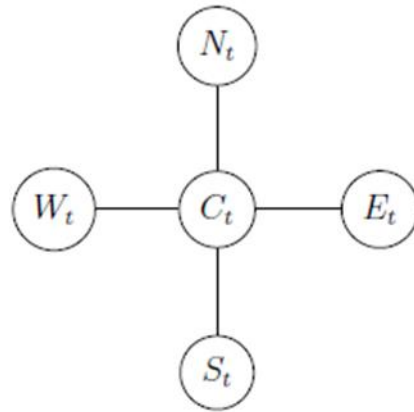# GPGPU HW3 report

R04943133
台大電子所
ICS 組
林岳徵

一開始我利用講義上提供的 Poisson editing eq.時

(a) The values to be solved.

(b) The corresponding target image.

$$C_b' = \frac{1}{4}\left[\underbrace{4C_t - (N_t + W_t + S_t + E_t) + (N_b + W_b)}_{\text{Fixed during iterations}} + \underbrace{(S_b + E_b)}_{\text{Current value}}\right]$$

實現了 `CalculateFixed` 和 `PoissonImageCloningIteration` 兩個函式

CalculateFixed

```
 8   __global__ void CalculateFixed(
 9          const float *background,
10          const float *target,
11          const float *mask,
12          float *output,
13          const int wb,
14          const int hb,
15          const int wt,
16          const int ht,
17          const int oy,
18          const int ox
19   )
20   {
21          const int xt = blockIdx.x * blockDim.x + threadIdx.x;
22          const int yt = blockIdx.y * blockDim.y + threadIdx.y;
23
24          //target array index for the current pixel location
25          const int curt = wt*yt + xt;//coordinate
26
27          if (yt < ht and xt < wt and mask[curt] > 127.0f) {//to simple clone
28                  return;
29          }
30          const int dir[4][2]{{1,1}, {1,-1}, {-1,1}, {-1,-1}};//definition direction
31          float temp[3];
32          temp[0] = 4*target[3*curt+0];
33          temp[1] = 4*target[3*curt+1];
34          temp[2] = 4*target[3*curt+2];

36          {       //declare neighbor coordinate
37                  const int nx = xt + dir[i][0];
38                  const int ny = yt + dir[i][0];
39                  const int curn = wt*ny + nx;
40                  if (nx >= 0 && ny >= 0 && nx < wt && ny < ht) {//inside boundry
41          temp[0] -= target[3*curn + 0];
42          temp[1] -= target[3*curn + 1];
43          temp[2] -= target[3*curn + 2];
44        } else {
45          temp[0] -= 255.0f;
46          temp[1] -= 255.0f;
47          temp[2] -= 255.0f;
48        }
49        if ((nx < 0 || ny < 0 || nx >= wt || ny >= ht) || mask[curn] < 127.0f) {//at edge
50                  const int bx = nx + ox;
51                  const int by = ny + oy;
52                  const int curb = (wb*by + bx)*3;
53                  temp[0] += background[curb+0];
54                  temp[1] += background[curb+1];
55                  temp[2] += background[curb+2];
56        }
57      }
58          output[3*curt+0] = temp[0];
59          output[3*curt+1] = temp[1];
60          output[3*curt+2] = temp[2];
61   }
```

PoissonImageCloningIteration

```
63   __global__ void PoissonImageCloningIteration(
64         const float *fixed,
65         const float *mask,
66         const float *target,
67         float *output,
68         const int wt,
69         const int ht
70   )
71   {
72         const int xt = blockDim.x * blockIdx.x + threadIdx.x;
73         const int yt = blockDim.y * blockIdx.y + threadIdx.y;
74         const int curt = wt*yt + xt;
75         if (xt >= wt || yt >= ht || mask[curt] < 127.0f) {
76               return;
77         }
78         float temp[3];
79   temp[0] = fixed[3*curt+0];
80   temp[1] = fixed[3*curt+1];
81   temp[2] = fixed[3*curt+2];
82   const int dir[4][2] = {{1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
83   for (int i = 0; i < 4; ++i) {
84     const int nx = xt + dir[i][0];
85     const int ny = yt + dir[i][1];
86     const int curn = wt*ny + nx;
87     if (nx >= 0 && nx < wt && ny >= 0 && ny < ht && mask[curn] > 127.0f) {
88       temp[0] += target[3*curn+0];
89       temp[1] += target[3*curn+1];
90       temp[2] += target[3*curn+2];
91     }
92   }
93   output[3*curt+0] = temp[0] / 4;
94   output[3*curt+1] = temp[1] / 4;
95   output[3*curt+2] = temp[2] / 4;
96  }
97
```

發現雖然到最後可以達到收斂，但是 iteration 的次數太多
因此我實現了講義上提供的加速方法 Successive Over-Relaxation method

$$C'_{b,SOR} = \omega C'_b + (1 - \omega)C_b.$$

```
98   //Implement  successive over-relaxation acceleration
99   __global__ void sor(
100          float w,
101          float *cur,
102          float *nxt,
103          const int wt,
104          const int ht
105   )
106   {
107          const int xt = blockIdx.x * blockDim.x + threadIdx.x;
108          const int yt = blockIdx.y * blockDim.y + threadIdx.y;
109          const int curt = wt*yt + xt;
110
111          nxt[curt*3] = nxt[curt*3] + (1-w)*cur[curt*3];
112          nxt[curt*3+1] = nxt[curt*3+1] + (1-w)*cur[curt*3+1];
113          nxt[curt*3+2] = nxt[curt*3+2] + (1-w)*cur[curt*3+1];
114   }
```

大幅度降低 iteration 的次數和需要的時間

計時的 timer 我是加在 SOR 開始前和 iteration 結束後

```
169          //timer start
170          Timer timer;
171          timer.Start();


211          //print timer
212          timer.Pause();
213          printf_timer(timer);
```

ω	SOR Iterations	Normal iterations	Execution Time(us)
1	10	20000	8269867
1.5	10	15000	6204464
2	10	9000	3723372
2.5	10	15000	6198683
3	10	20000	8262666

由跑出來的 data，我們可以發現當ω 越大的時候，執行時間會減少，以 2 為分水嶺，越大反而不會收斂，另外由於我們觀察影像是用肉眼去看，很難定義一個標準叫做完全縫合，所以多少會造成一些誤差。