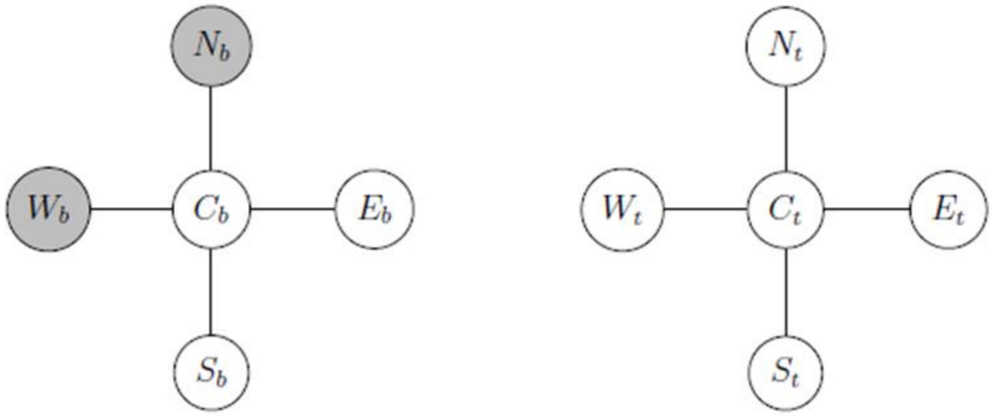
GPGPU HW3 report

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
下圖為實現後的 output.png

CalculateFixed

```
__global__ void CalculateFixed(
        const float *background,
        const float *target,
        const float *mask,
        float *output,
        const int wb,
        const int hb,
        const int wt,
        const int ht,
        const int oy,
        const int ox
)
{
        const int xt = blockIdx.x * blockDim.x + threadIdx.x;
        const int yt = blockIdx.y * blockDim.y + threadIdx.y;

        //target array index for the current pixel location
        const int curt = wt*yt + xt;//coordinate

        if (yt < ht and xt < wt and mask[curt] > 127.0f) {//to simple clone
                return;
        }
        const int dir[4][2]{{1,1}, {1,-1}, {-1,1}, {-1,-1}};//definition direction
        float temp[3];
        temp[0] = 4*target[3*curt+0];
        temp[1] = 4*target[3*curt+1];
        temp[2] = 4*target[3*curt+2];
        {       //declare neighbor coordinate
                const int nx = xt + dir[i][0];
                const int ny = yt + dir[i][0];
                const int curn = wt*ny + nx;
                if (nx >= 0 && ny >= 0 && nx < wt && ny < ht) {//inside boundry
            temp[0] -= target[3*curn + 0];
            temp[1] -= target[3*curn + 1];
            temp[2] -= target[3*curn + 2];
        } else {
            temp[0] -= 255.0f;
            temp[1] -= 255.0f;
            temp[2] -= 255.0f;
        }
        if ((nx < 0 || ny < 0 || nx >= wt || ny >= ht) || mask[curn] < 127.0f) {//at edge
                const int bx = nx + ox;
                const int by = ny + oy;
                const int curb = (wb*by + bx)*3;
                temp[0] += background[curb+0];
                temp[1] += background[curb+1];
                temp[2] += background[curb+2];
        }
    }
        output[3*curt+0] = temp[0];
        output[3*curt+1] = temp[1];
        output[3*curt+2] = temp[2];
}
```

PoissonImageCloningIteration

```
__global__ void PoissonImageCloningIteration(
        const float *fixed,
        const float *mask,
        const float *target,
        float *output,
        const int wt,
        const int ht
)
{
        const int xt = blockDim.x * blockIdx.x + threadIdx.x;
        const int yt = blockDim.y * blockIdx.y + threadIdx.y;
        const int curt = wt*yt + xt;
        if (xt >= wt || yt >= ht || mask[curt] < 127.0f) {
                return;
        }
        float temp[3];
    temp[0] = fixed[3*curt+0];
    temp[1] = fixed[3*curt+1];
    temp[2] = fixed[3*curt+2];
    const int dir[4][2] = {{1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
    for (int i = 0; i < 4; ++i) {
      const int nx = xt + dir[i][0];
      const int ny = yt + dir[i][1];
      const int curn = wt*ny + nx;
      if (nx >= 0 && nx < wt && ny >= 0 && ny < ht && mask[curn] > 127.0f) {
        temp[0] += target[3*curn+0];
        temp[1] += target[3*curn+1];
        temp[2] += target[3*curn+2];
      }
    }
    output[3*curt+0] = temp[0] / 4;
    output[3*curt+1] = temp[1] / 4;
    output[3*curt+2] = temp[2] / 4;
}
```

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

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

```
//Implement  successive over-relaxation acceleration
__global__ void sor(
        float w,
        float *cur,
        float *nxt,
        const int wt,
        const int ht
)
{
        const int xt = blockIdx.x * blockDim.x + threadIdx.x;
        const int yt = blockIdx.y * blockDim.y + threadIdx.y;
        const int curt = wt*yt + xt;

        nxt[curt*3] = nxt[curt*3] + (1-w)*cur[curt*3];
        nxt[curt*3+1] = nxt[curt*3+1] + (1-w)*cur[curt*3+1];
        nxt[curt*3+2] = nxt[curt*3+2] + (1-w)*cur[curt*3+1];
}
```

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

```
        //timer start
        Timer timer;
        timer.Start();
```

```
        //print timer
        timer.Pause();
        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。