

代码优化

介绍

- 图像处理中存在很多函数，可以对这些函数进行优化
- 本实验主要关注两种图像处理操作
- 旋转：对图像逆时针旋转90度
- 平滑：对图像进行模糊操作
- 图像用二维矩阵 M 表示， M_{ij} 表示图像 M 的第 (i, j) 像素的值，像素值用红，绿，蓝表示

介绍

- 旋转操作作用下面2个操作表示
 - Transpose: 对第 (i,j) 个像素对, M_{ij} 和 M_{ji} 交换
 - Exchange rows: 行 i 和行 $N-1-i$ 交换

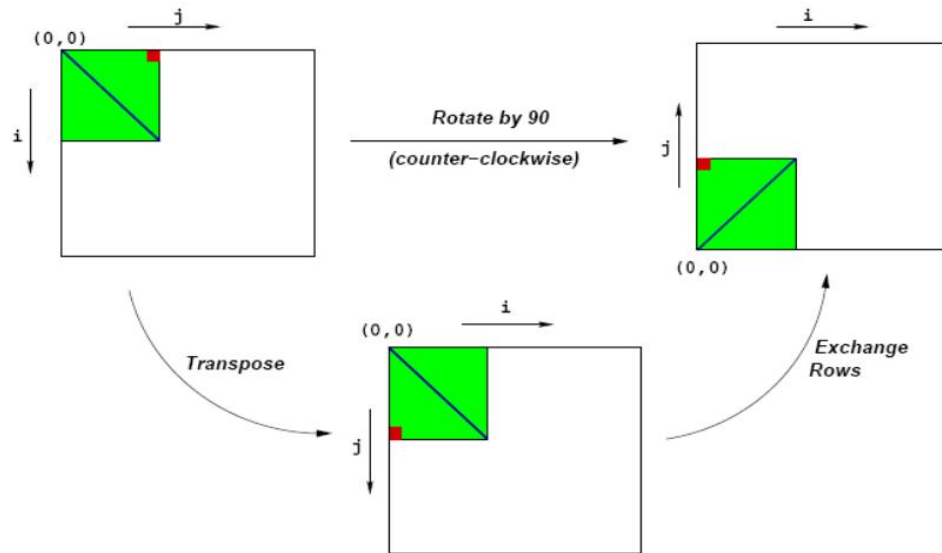


Figure 1: Rotation of an image by 90° counterclockwise

- 平滑操作：每个像素用周围像素值的平均值表示
- 像素 $M2[1][1]$ 和 $M2[N-1][N-1]$ 如下图所示

$$M2[1][1] = \frac{\sum_{i=0}^2 \sum_{j=0}^2 M1[i][j]}{9}$$

$$M2[N-1][N-1] = \frac{\sum_{i=N-2}^{N-1} \sum_{j=N-2}^{N-1} M1[i][j]}{4}$$

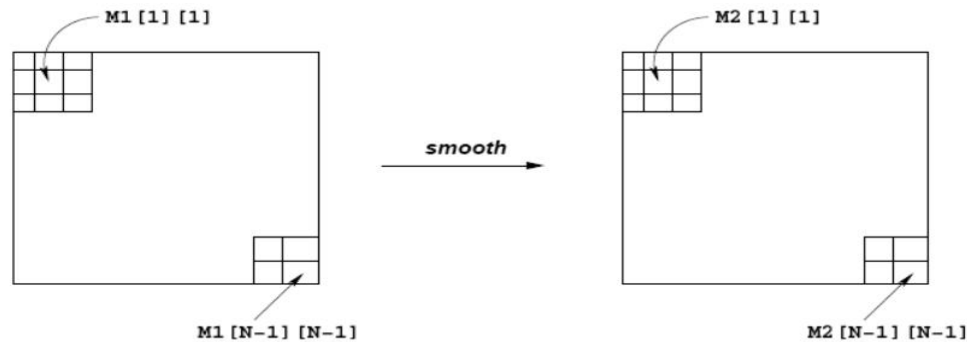


Figure 2: Smoothing an image

步骤

- `tar xvf perflab-handout.tar`
- `kernels.c`: 唯一需要修改和提交的文件
- `driver.c`: 评价修改的`kernels.c`性能
- `make driver`
- `./driver`

步骤

- typedef struct {
- unsigned short red; /* R value */
- unsigned short green; /* G value */
- unsigned short blue; /* B value */
- } pixel
- 图像用一维数组表示，第 (i,j) 个像素表示为 $I[RIDX(i,j,n)]$ ， n 为图像的维数
- #define RIDX(i,j,n) ((i)*(n)+(j))

步骤

- 旋转操作

```
void naive_rotate(int dim, pixel *src, pixel *dst) {  
    int i, j;  
    for(i=0; i < dim; i++)  
        for(j=0; j < dim; j++)  
            dst[RIDX(dim-1-j,i,dim)] = src[RIDX(i,j,dim)];  
    return;  
}
```

目标：使用代码优化技术使旋转操作运行的更快

步骤

- 平滑操作

```
void naive_smooth(int dim, pixel *src, pixel *dst) {  
    int i, j;  
    for(i=0; i < dim; i++)  
        for(j=0; j < dim; j++)  
            dst[RIDX(i,j,dim)] = avg(dim, i, j, src); /* Smooth the (i,j)th  
pixel */  
    return;  
}
```

目标： 使用代码优化技术使平滑操作运行的更快

性能评判方法

- CPE or Cycles per Element
- 如果一个函数使用C个周期去执行一个大小为N*N的图像，那么 $CPE = C / (N * N)$ ，因此CPE越小越好

如何定义函数

- 在**kernel.c**中有**rotate**的实现方法，我们需要添加自己的**rotate**实现，函数名自己命名，然后通过

```
void register_rotate_functions() {  
    add_rotate_function(&rotate, rotate_descr);  
    add_rotate_function(&my_rotate, my_rotate_descr) ;  
}
```

- 把自己实现的函数注册进去

编码规则

- 只能用**ANSI C**，不能用嵌入式汇编
- 不能修改测量时间的机制（**CPE**）
- 只能修改**kernels.c**，可以定义宏，全局变量，函数