系统程序设计作业Unit6

2014302580341 卓越二班 余璞轩

1.问题描述

像素的结构为

```
typedef struct {
   unsigned short red   : 8;
   unsigned short green : 8;
   unsigned short blue  : 8;
   unsigned short alpha : 8;
} pixel;
```

每个像素32位，即4字节。

缓存大小为2^{14}字节，一个缓存行2^5字节，所以一共有2^9行，每行存储8个像素。

2.rotate函数

```
char rotate_descr[] = "Naive Row-wise Traversal of src";
void rotate(int dim, pixel *src, pixel *dst) {
    int i, j;
    for(i=0; i < dim; i++) {
        for(j=0; j < dim; j++) {
            COPY(&dst[PIXEL(dim-1-j,i,dim)], &src[PIXEL(i,j,dim)]);
        }
    }
    return;
}
```

分析一下原来的函数。`src`的命中率很正常，但是`dst`的命中率为0。写入第一列固然全部不命中，但是可以将后续的列载入缓存，这样后续的列就可以命中。

想到将整个图像，划分成若干个块分别进行旋转。

但是还有一个问题，就是题目要求将图像顺时针旋转，如果就这么从上到下扫描的话，`dst`数组总是先填充最右边，然后开始向左。这样对提高命中率没有任何帮助。如果反过来，`src`从下往上扫描的话，`dst`数组就是从左到右填充，就可以满足我们的需求了。

代码如下：

```
char optimized_descr[] = "Optimized rotation";
void optimized_rotate(int dim, pixel *src, pixel *dst) {
    int block_size = 4;
    int i, j, k, l;
    for (l = 0; l < dim; l += block_size)
        for (k = 0; k < dim; k += block_size)
            for (j = l + block_size - 1; j >= l; j--)
                for (i = k; i < k + block_size; i++)
                    COPY(&dst[PIXEL(dim-1-j,i,dim)], &src[PIXEL(i,j,dim
)]);
}
```

3.smooth函数

这个函数的修改比`rotate`要好想一些。就是修改程序局部性的基本想法：交换循环的顺序。

先行后列的规则显然比默认的先列后行要高效的多。

代码如下：

```
char opt_descr[] = "Optimized smooth";
void optimized_smooth(int dim, pixel *src, pixel *dst) {
    int i, j;
    for(i=0; i<dim;i++) {
        COPY(&dst[PIXEL(i,0,dim)], &src[PIXEL(i,0,dim)]);
        COPY(&dst[PIXEL(i,dim-1,dim)], &src[PIXEL(i,dim-1,dim)]);
    }
    for(j=1; j<dim-1;j++) {
        COPY(&dst[PIXEL(0,j,dim)], &src[PIXEL(0,j,dim)]);
```

```
        COPY(&dst[PIXEL(dim-1,j,dim)], &src[PIXEL(dim-1,j,dim)]);
    }
    for(j=1; j<dim-1; j++) {
        for(i=1; i<dim-1; i++) {
            SMOOTH(&dst[PIXEL(j,i,dim)],
                   &src[PIXEL(j,i,dim)],
                   &src[PIXEL(j-1,i,dim)],
                   &src[PIXEL(j+1,i,dim)],
                   &src[PIXEL(j,i+1,dim)],
                   &src[PIXEL(j,i-1,dim)],
                   &src[PIXEL(j-1,i-1,dim)],
                   &src[PIXEL(j+1,i+1,dim)],
                   &src[PIXEL(j-1,i+1,dim)],
                   &src[PIXEL(j+1,i-1,dim)]);
        }
    }
    return;
}
```

4.结果

编译所有文件，执行`./a.out`，得到结果如下：

```
Rotate: Version = Rotate Reference Naive Implementation!:
Dim 64  128 256 512 1024    Mean
hitrate 86.8    43.8    43.8    43.7    43.7
Incr.   1.00    1.00    1.00    1.00    1.00    1.00

Rotate: Version = Naive Row-wise Traversal of src:
Dim 64  128 256 512 1024    Mean
hitrate 86.8    43.8    43.8    43.7    43.7
Incr.   1.00    1.00    1.00    1.00    1.00    1.00

Rotate: Version = Optimized rotation:
Dim 64  128 256 512 1024    Mean
hitrate 87.1    81.2    81.2    80.9    81.0
Incr.   1.00    1.86    1.86    1.85    1.85    1.64

Best algo here: Optimized rotation,     1.640945


Smooth: Version = Smooth Reference Naive Implementation!:
Dim 64  128 256 512 1024    Mean
hitrate 63.1    45.4    45.6    45.7    45.8
Incr.   1.00    1.00    1.00    1.00    1.00    1.00
```

```
Smooth: Version = SMOOTH: Naive Row-wise Traversal of src:
Dim 64  128 256 512 1024    Mean
hitrate 63.1    45.4    45.6    45.7    45.8
Incr.   1.00    1.00    1.00    1.00    1.00    1.00

Smooth: Version = Optimized smooth:
Dim 64  128 256 512 1024    Mean
hitrate 63.1    63.8    64.2    64.4    64.5
Incr.   1.00    1.41    1.41    1.41    1.41    1.31

Best algo here: Optimized smooth,   1.314475
```

看的出来修改过的算法比之前有明显的效率提升。