Lab3 图像优化实验

李仁杰 13307130279

一、实验目的

1. 理解编译器和 cache 对程序速度的影响；
2. 学习程序优化，提高代码能力。

二、实验内容

这个实验是一个图像优化实验，需要大家将 perflab-handout.tar 中的代码文件 kernels.c 中的 rotate、smooth 这两个函数进行优化。（利用第五章学习的优化程序性能的知识）。

三、实验过程

1. rotate 函数

```c
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)];
}
```

首先我们找到了 rotate 函数中直接调用的 naïve_rotate，可以发现，这里频繁调用 RIDX(dim-1,j,I,dim) 和 RIDX(I,j,dim)，可以在头文件 defs.h 中找到宏定义：#define RIDX(i,j,n) ((i)*(n)+(j))。结合 perflab.pdf 中的指示，不难发现，这个可以理解为一幅画的旋转，它将将所有的像素进行行列调位、导致整幅图画进行了 90 度旋转。（如下图）

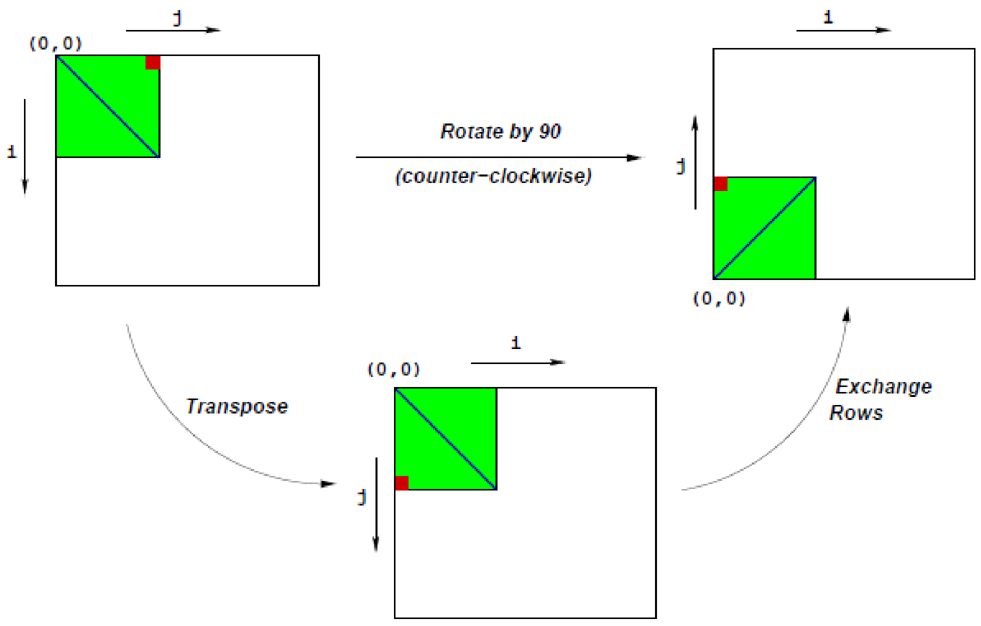


Figure 1: Rotation of an image by 90° counterclockwise

由于实际上各个像素点的访问是临近的，不需要每次都进行定位。因此这种重复的位置计算没必要保留。最终采用的方案是直接以指针的操作来实现像素间跳转。

之后，考虑到流水线的结构，采用分部展开的方式。通过若干次尝试，发现 32 路展开的效果最好。至于为什么 32 路展开的效果最好，应该是因为 Dim 的大小分别为 64,128,256,512,1024，这些数据均为 32 的倍数。因此，我们考虑到 cache 的大小，采用分块策略， 每一个块大小为 32，这样可以做到 cache 友好，大幅度提高程序的效率。

2. smooth 函数

```
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);
}
```

这里也是先找出 naïve_smooth 函数，发现不止运用了 RIDX(I,j,dim)的宏，还包括一个 avg(dim,i,j,src)的函数，我们将这其中的函数找出：

```
static pixel avg(int dim, int i, int j, pixel *src)
{
    int ii, jj;
    pixel_sum sum;
    pixel current_pixel;

    initialize_pixel_sum(&sum);
    for(ii = max(i-1, 0); ii <= min(i+1, dim-1); ii++)
    for(jj = max(j-1, 0); jj <= min(j+1, dim-1); jj++)
        accumulate_sum(&sum, src[RIDX(ii, jj, dim)]);

    assign_sum_to_pixel(&current_pixel, sum);
    return current_pixel;
}
```

不难看出，这个程序的返回值是 assign_sum_to_pixel()函数观察其函数体可以发现，这是返回一个结构类型的值。另外这里的二重循环中又在频繁调用 accumulate_sum()，min(),max()这三个函数。

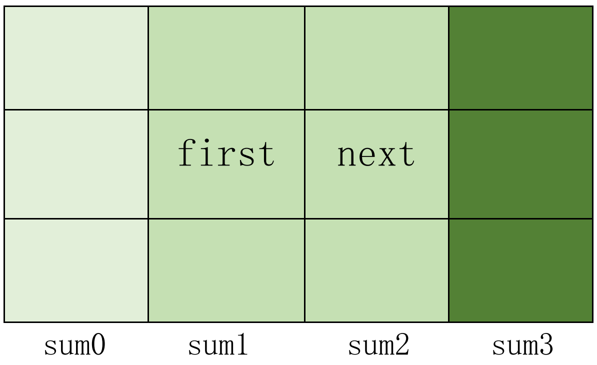
```
/* Compute min and max of two integers, respectively */
static int min(int a, int b) { return (a < b ? a : b); }
static int max(int a, int b) { return (a > b ? a : b); }

static void accumulate_sum(pixel_sum *sum, pixel p)
{
    sum->red += (int) p.red;
    sum->green += (int) p.green;
    sum->blue += (int) p.blue;
    sum->num++;
    return;
}
```

于是我们把这三个函数都找出来，发现 min()和 max()只是简单的返回函数的最小值和最大值，在 avg()函数中是用来判断计算平均值的方格是否超出 dim 标记的正方形。这里 avg(dim,i,j)计算的是以 src[i][j]为中心的一个九宫格中的 red，green，blue 三原色的平均值。我们可以设想，只有在 src[][]的四个角和四条边上，这里的 min()和 max()的判断才会有用，于是我们将四个角和四条边特殊处理。在做过这个预处理之后，把所有的函数都集成到 smooth()函数中，这样就不会再出现频繁调用函数的现象。

之后我们发现，这个程序中还存在着重复计算的现象。如下图，在计算

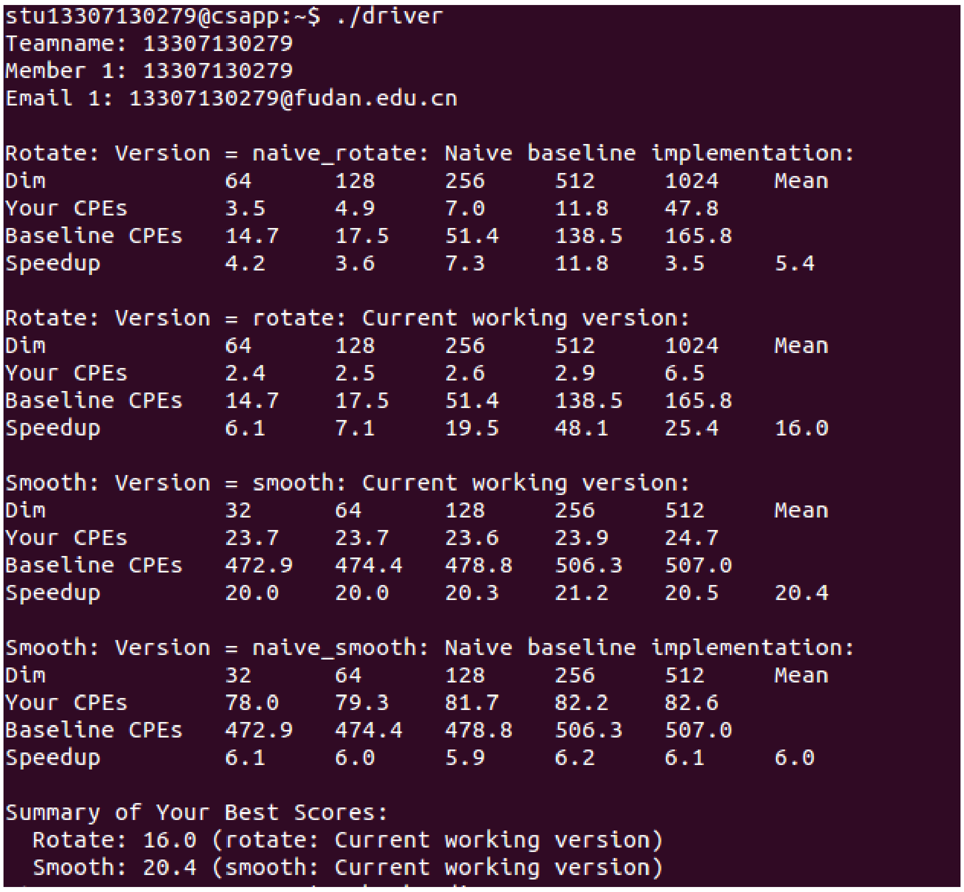
first 区域的 avg()值的时候，程序会对其周围的九宫格进行加和，于是途中浅绿色的地方在计算 next 的时候会被再次计算。于是这里我们引入一个对每三列的结果进行储存的随机变量 sum0，sum1，sum2，sum3。在计算下一个位置，也就是 next 的时候，我们进行维护 sum0=sum1,sum1=sum2,sum2 重新进行加和，做出 sum3 的格式。这样，计算的次数大规模减少。



接下来，我们可以发现，在计算一列的结果的时候，下一列的值也会发生重复计算，于是我们引入 sum4 和 sum5，采用同样的方法，一次计算两列。最后，采取分步展开的方法，最终确定了四路展开的效果最好。（在四路之后随着展开的路数增多，速度并没有明显提高）

四、实验结果

键入命令 make clean 先把所有的.o 文件删除，接着运用 make driver 命令重新编译程序，最后用./driver 命令运行程序，得到结果如下图。可以看到，对于 rotate 函数，优化达到了 16.0，而对于 smooth 函数，优化达到了 20.4，还算是一个不错的成果。



```
stu13307130279@csapp:~$ ./driver
Teamname: 13307130279
Member 1: 13307130279
Email 1: 13307130279@fudan.edu.cn

Rotate: Version = naive_rotate: Naive baseline implementation:
Dim             64      128     256     512     1024    Mean
Your CPEs       3.5     4.9     7.0     11.8    47.8
Baseline CPEs   14.7    17.5    51.4    138.5   165.8
Speedup         4.2     3.6     7.3     11.8    3.5     5.4

Rotate: Version = rotate: Current working version:
Dim             64      128     256     512     1024    Mean
Your CPEs       2.4     2.5     2.6     2.9     6.5
Baseline CPEs   14.7    17.5    51.4    138.5   165.8
Speedup         6.1     7.1     19.5    48.1    25.4    16.0

Smooth: Version = smooth: Current working version:
Dim             32      64      128     256     512     Mean
Your CPEs       23.7    23.7    23.6    23.9    24.7
Baseline CPEs   472.9   474.4   478.8   506.3   507.0
Speedup         20.0    20.0    20.3    21.2    20.5    20.4

Smooth: Version = naive_smooth: Naive baseline implementation:
Dim             32      64      128     256     512     Mean
Your CPEs       78.0    79.3    81.7    82.2    82.6
Baseline CPEs   472.9   474.4   478.8   506.3   507.0
Speedup         6.1     6.0     5.9     6.2     6.1     6.0

Summary of Your Best Scores:
  Rotate: 16.0 (rotate: Current working version)
  Smooth: 20.4 (smooth: Current working version)
```

五、收获与反思

通过本次实验，我理解了代码优化的一些通用方法，例如减少函数的反复调用，提前计算要利用的值，循环展开，以及提高 cache 利用率。这些能力为以后编写大型程序做了很好的铺垫。

这次实验的文件也会在截止日期 5 月 10 日 12 点之后在我的 github 上开源。地址 https://github.com/lirenjie95/CSAPP

附录

kernels.c 中的 rotate 和 smooth 段

```
char rotate_descr[] = "rotate: Current working version";
void rotate(int dim, pixel *src, pixel *dst)
{
    int i,j;
    dst += dim*(dim-1);// Move to the last bvisk
    for(i = 0;i < dim;i += 32)
    {
        for(j = 0;j < dim;j++)
        {
            *dst = *src; src += dim; dst++;//1
            *dst = *src; src += dim; dst++;//2
            *dst = *src; src += dim; dst++;//3
            *dst = *src; src += dim; dst++;//4
            *dst = *src; src += dim; dst++;//5
            *dst = *src; src += dim; dst++;//6
            *dst = *src; src += dim; dst++;//7
            *dst = *src; src += dim; dst++;//8
            *dst = *src; src += dim; dst++;//9
            *dst = *src; src += dim; dst++;//10
            *dst = *src; src += dim; dst++;//11
            *dst = *src; src += dim; dst++;//12
            *dst = *src; src += dim; dst++;//13
            *dst = *src; src += dim; dst++;//14
            *dst = *src; src += dim; dst++;//15
            *dst = *src; src += dim; dst++;//16
            *dst = *src; src += dim; dst++;//17
            *dst = *src; src += dim; dst++;//18
            *dst = *src; src += dim; dst++;//19
            *dst = *src; src += dim; dst++;//20
            *dst = *src; src += dim; dst++;//21
            *dst = *src; src += dim; dst++;//22
            *dst = *src; src += dim; dst++;//23
            *dst = *src; src += dim; dst++;//24
            *dst = *src; src += dim; dst++;//25
            *dst = *src; src += dim; dst++;//26
            *dst = *src; src += dim; dst++;//27
            *dst = *src; src += dim; dst++;//28
            *dst = *src; src += dim; dst++;//29
            *dst = *src; src += dim; dst++;//30
            *dst = *src; src += dim; dst++;//31
            *dst = *src; src++;
            src -= (dim<<5)-dim;// src = src-dim*31
            dst -= 31+dim;// Find the next row or column.
        }
        dst += dim*dim;dst += 32;
        src += (dim<<5)-dim;// Reset
    }
}
char smooth_descr[] = "smooth: Current working version";
void smooth(int dim, pixel *src, pixel *dst)
{
    int i,j;
    // Four corners: only consider four blocks.
    dst[0].red = (src[0].red+src[1].red+src[dim].red+src[dim+1].red)>>2;
    dst[0].blue = (src[0].blue+src[1].blue+src[dim].blue+src[dim+1].blue)>>2;
    dst[0].green =
(src[0].green+src[1].green+src[dim].green+src[dim+1].green)>>2;
    //corner[0][0]
    dst[dim-1].red = (src[dim-1].red+src[dim-2].red+src[dim*2-1].red+src[dim*2-
2].red)>>2;
    dst[dim-1].blue = (src[dim-1].blue+src[dim-2].blue+src[dim*2-
1].blue+src[dim*2-2].blue)>>2;
    dst[dim-1].green = (src[dim-1].green+src[dim-2].green+src[dim*2-
1].green+src[dim*2-2].green)>>2;
    //corner[0][dim-1]
    dst[dim*(dim-1)].red = (src[dim*(dim-1)].red+src[dim*(dim-
1)+1].red+src[dim*(dim-2)].red+src[dim*(dim-2)+1].red)>>2;
    dst[dim*(dim-1)].blue = (src[dim*(dim-1)].blue+src[dim*(dim-
1)+1].blue+src[dim*(dim-2)].blue+src[dim*(dim-2)+1].blue)>>2;
    dst[dim*(dim-1)].green = (src[dim*(dim-1)].green+src[dim*(dim-
1)+1].green+src[dim*(dim-2)].green+src[dim*(dim-2)+1].green)>>2;
    //cornor[dim-1][0]
    dst[dim*dim-1].red = (src[dim*dim-1].red+src[dim*dim-2].red+src[dim*(dim-1)-
1].red+src[dim*(dim-1)-2].red)>>2;
    dst[dim*dim-1].blue = (src[dim*dim-1].blue+src[dim*dim-2].blue+src[dim*(dim-
1)-1].blue+src[dim*(dim-1)-2].blue)>>2;
    dst[dim*dim-1].green = (src[dim*dim-1].green+src[dim*dim-
2].green+src[dim*(dim-1)-1].green+src[dim*(dim-1)-2].green)>>2;
    //corner[dim-1][dim-1]
    int pos = dim-1;//Four sides
    for(i=1;i<pos;i++)
    {
        dst[i].red = (src[i].red+src[i-
1].red+src[i+1].red+src[i+dim].red+src[i+1+dim].red+src[i-1+dim].red)/6;
        dst[i].green = (src[i].green+src[i-
1].green+src[i+1].green+src[i+dim].green+src[i+1+dim].green+src[i-
1+dim].green)/6;
        dst[i].blue = (src[i].blue+src[i-
1].blue+src[i+1].blue+src[i+dim].blue+src[i+1+dim].blue+src[i-1+dim].blue)/6;
    }//row[0]
    pos = dim*dim-1;
    for(i=dim*(dim-1)+1;i<pos;i++)
    {
        dst[i].red = (src[i].red+src[i-1].red+src[i+1].red+src[i-
dim].red+src[i+1-dim].red+src[i-1-dim].red)/6;
        dst[i].green = (src[i].green+src[i-1].green+src[i+1].green+src[i-
dim].green+src[i+1-dim].green+src[i-1-dim].green)/6;
        dst[i].blue = (src[i].blue+src[i-1].blue+src[i+1].blue+src[i-
dim].blue+src[i+1-dim].blue+src[i-1-dim].blue)/6;
    }//row[dim-1]
    pos = dim*dim-dim;
    for(i=dim;i<pos;i+=dim)
    {
        dst[i].red = (src[i].red+src[i-
dim].red+src[i+1].red+src[i+dim].red+src[i+1+dim].red+src[i-dim+1].red)/6;
        dst[i].green = (src[i].green+src[i-
dim].green+src[i+1].green+src[i+dim].green+src[i+1+dim].green+src[i-
dim+1].green)/6;
        dst[i].blue = (src[i].blue+src[i-
dim].blue+src[i+1].blue+src[i+dim].blue+src[i+1+dim].blue+src[i-dim+1].blue)/6;
    }//column[0]
    pos = dim*dim-1;
    for(i=dim+dim-1;i<pos;i+=dim)
    {
        dst[i].red = (src[i].red+src[i-1].red+src[i-
dim].red+src[i+dim].red+src[i-dim-1].red+src[i-1+dim].red)/6;
        dst[i].green = (src[i].green+src[i-1].green+src[i-
dim].green+src[i+dim].green+src[i-dim-1].green+src[i-1+dim].green)/6;
        dst[i].blue = (src[i].blue+src[i-1].blue+src[i-
dim].blue+src[i+dim].blue+src[i-dim-1].blue+src[i-1+dim].blue)/6;
    }//column[dim-1]
    /*-------------------------So, we have solved all special points.-----------
-------------------------------------*/
    pixel *vis1=&src[0];//src[0][0]
    pixel *vis2=&src[dim];//src[1][0]
    pixel *vis3=&src[dim+dim];//src[2][0]
    pixel *vis4=&src[dim+dim+dim];//src[3][0]
    int sum0_red,sum0_green,sum0_blue;
    int sum1_red,sum1_green,sum1_blue;
    int sum2_red,sum2_green,sum2_blue;
    int sum3_red,sum3_green,sum3_blue;
    int sum4_red,sum4_green,sum4_blue;
    int sum5_red,sum5_green,sum5_blue;
    int first=dim+1;
    int next=first+dim;// Next row of 'first'
    for(i=1;i<dim-2;i+=2)// Two rows for one time
    {
        sum0_red=vis2->red;     sum0_blue=vis2->blue;  sum0_green=vis2->green;
        sum0_red+=vis3->red;    sum0_blue+=vis3->blue; sum0_green+=vis3->green;
        sum3_red=sum0_red+vis4->red;
        sum3_green=sum0_green+vis4->green;
        sum3_blue=sum0_blue+vis4->blue;
        sum0_red+=vis1->red; sum0_blue+=vis1->blue;    sum0_green+=vis1->green;
        vis1++;vis2++;vis3++;vis4++;
        //First block
        sum1_red=vis2->red;      sum1_blue=vis2->blue; sum1_green=vis2->green;
        sum1_red+=vis3->red;    sum1_blue+=vis3->blue; sum1_green+=vis3->green;
        sum4_red=sum1_red+vis4->red;
        sum4_green=sum1_green+vis4->green;
        sum4_blue=sum1_blue+vis4->blue;
        sum1_red+=vis1->red; sum1_blue+=vis1->blue;    sum1_green+=vis1->green;
        vis1++;vis2++;vis3++;vis4++;
        //Next block
        sum2_red=vis2->red;      sum2_blue=vis2->blue; sum2_green=vis2->green;
        sum2_red+=vis3->red;    sum2_blue+=vis3->blue; sum2_green+=vis3->green;
        sum5_red=sum2_red+vis4->red;
        sum5_green=sum2_green+vis4->green;
        sum5_blue=sum2_blue+vis4->blue;
        sum2_red+=vis1->red; sum2_blue+=vis1->blue;    sum2_green+=vis1->green;
        vis1++;vis2++;vis3++;vis4++;
        //Third bvisk
        dst[first].red=(sum0_red+sum1_red+sum2_red)/9;
        dst[first].blue=(sum0_blue+sum1_blue+sum2_blue)/9;
        dst[first].green=(sum0_green+sum1_green+sum2_green)/9;
        first++;
        // Make the first row.
        dst[next].red=(sum3_red+sum4_red+sum5_red)/9;
        dst[next].blue=(sum3_blue+sum4_blue+sum5_blue)/9;
        dst[next].green=(sum3_green+sum4_green+sum5_green)/9;
        next++;
        // Make the next row.
        sum0_red=sum1_red;        sum1_red=sum2_red;
        sum0_green=sum1_green;    sum1_green=sum2_green;
        sum0_blue=sum1_blue;  sum1_blue=sum2_blue;
        sum3_red=sum4_red;        sum4_red=sum5_red;
        sum3_green=sum4_green;    sum4_green=sum5_green;
        sum3_blue=sum4_blue;  sum4_blue=sum5_blue;
        // Save these to calculate the next.
        for(j=2;j<dim-4;j+=4)
        {
            sum2_red=vis2->red;      sum2_blue=vis2->blue;
    sum2_green=vis2->green;
            sum2_red+=vis3->red;    sum2_blue+=vis3->blue;
    sum2_green+=vis3->green;
            sum5_red=sum2_red+vis4->red;
            sum5_green=sum2_green+vis4->green;
            sum5_blue=sum2_blue+vis4->blue;
            sum2_red+=vis1->red;  sum2_blue+=vis1->blue;
    sum2_green+=vis1->green;
            vis1++;vis2++;vis3++;vis4++;
            //First bvisk
            dst[first].red=((sum0_red+sum1_red+sum2_red)/9);
            dst[first].blue=((sum0_blue+sum1_blue+sum2_blue)/9);
            dst[first].green=((sum0_green+sum1_green+sum2_green)/9);
            first++;
            // Make the first row.
            dst[next].red=((sum3_red+sum4_red+sum5_red)/9);
            dst[next].blue=((sum3_blue+sum4_blue+sum5_blue)/9);
            dst[next].green=((sum3_green+sum4_green+sum5_green)/9);
            next++;
            // Make the next row.
            sum0_red=sum1_red;        sum1_red=sum2_red;
            sum0_green=sum1_green;    sum1_green=sum2_green;
            sum0_blue=sum1_blue;  sum1_blue=sum2_blue;
            sum3_red=sum4_red;        sum4_red=sum5_red;
            sum3_green=sum4_green;    sum4_green=sum5_green;
            sum3_blue=sum4_blue;  sum4_blue=sum5_blue;
            // Save these to calculate the next.
            sum2_red=vis2->red;      sum2_blue=vis2->blue;
    sum2_green=vis2->green;
            sum2_red+=vis3->red;    sum2_blue+=vis3->blue;
    sum2_green+=vis3->green;
```

```
          sum5_red=sum2_red+vis4->red;
          sum5_green=sum2_green+vis4->green;
          sum5_blue=sum2_blue+vis4->blue;
          sum2_red+=vis1->red;  sum2_blue+=vis1->blue;
sum2_green+=vis1->green;
          vis1++;vis2++;vis3++;vis4++;
          dst[first].red=((sum0_red+sum1_red+sum2_red)/9);
          dst[first].blue=((sum0_blue+sum1_blue+sum2_blue)/9);
          dst[first].green=((sum0_green+sum1_green+sum2_green)/9);
          first++;
          dst[next].red=((sum3_red+sum4_red+sum5_red)/9);
          dst[next].blue=((sum3_blue+sum4_blue+sum5_blue)/9);
          dst[next].green=((sum3_green+sum4_green+sum5_green)/9);
          next++;
          sum0_red=sum1_red;        sum1_red=sum2_red;
          sum0_green=sum1_green;    sum1_green=sum2_green;
          sum0_blue=sum1_blue;  sum1_blue=sum2_blue;
          sum3_red=sum4_red;        sum4_red=sum5_red;
          sum3_green=sum4_green;    sum4_green=sum5_green;
          sum3_blue=sum4_blue;  sum4_blue=sum5_blue;
          // 2 road
          sum2_red=vis2->red;      sum2_blue=vis2->blue;
sum2_green=vis2->green;
          sum2_red+=vis3->red;  sum2_blue+=vis3->blue;
sum2_green+=vis3->green;
          sum5_red=sum2_red+vis4->red;
          sum5_green=sum2_green+vis4->green;
          sum5_blue=sum2_blue+vis4->blue;
          sum2_red+=vis1->red;  sum2_blue+=vis1->blue;
sum2_green+=vis1->green;
          vis1++;vis2++;vis3++;vis4++;
          dst[first].red=((sum0_red+sum1_red+sum2_red)/9);
          dst[first].blue=((sum0_blue+sum1_blue+sum2_blue)/9);
          dst[first].green=((sum0_green+sum1_green+sum2_green)/9);
          first++;
          dst[next].red=((sum3_red+sum4_red+sum5_red)/9);
          dst[next].blue=((sum3_blue+sum4_blue+sum5_blue)/9);
          dst[next].green=((sum3_green+sum4_green+sum5_green)/9);
          next++;
          sum0_red=sum1_red;        sum1_red=sum2_red;
          sum0_green=sum1_green;    sum1_green=sum2_green;
          sum0_blue=sum1_blue;  sum1_blue=sum2_blue;
          sum3_red=sum4_red;        sum4_red=sum5_red;
          sum3_green=sum4_green;    sum4_green=sum5_green;
          sum3_blue=sum4_blue;  sum4_blue=sum5_blue;
          // 3 road
          sum2_red=vis2->red;   sum2_blue=vis2->blue; sum2_green=vis2->green;
          sum2_red+=vis3->red;  sum2_blue+=vis3->blue;
sum2_green+=vis3->green;
          sum5_red=sum2_red+vis4->red;
          sum5_green=sum2_green+vis4->green;
          sum5_blue=sum2_blue+vis4->blue;
          sum2_red+=vis1->red;  sum2_blue+=vis1->blue;
sum2_green+=vis1->green;
          vis1++;vis2++;vis3++;vis4++;
         dst[first].red=((sum0_red+sum1_red+sum2_red)/9);
         dst[first].blue=((sum0_blue+sum1_blue+sum2_blue)/9);
         dst[first].green=((sum0_green+sum1_green+sum2_green)/9);
         first++;
          dst[next].red=((sum3_red+sum4_red+sum5_red)/9);
          dst[next].blue=((sum3_blue+sum4_blue+sum5_blue)/9);
          dst[next].green=((sum3_green+sum4_green+sum5_green)/9);
          next++;
         sum0_red=sum1_red;         sum1_red=sum2_red;
          sum0_green=sum1_green;    sum1_green=sum2_green;
          sum0_blue=sum1_blue;  sum1_blue=sum2_blue;
          sum3_red=sum4_red;        sum4_red=sum5_red;
          sum3_green=sum4_green;    sum4_green=sum5_green;
          sum3_blue=sum4_blue;  sum4_blue=sum5_blue;
          // 4 road
     }
     for(;j<dim-1;j++)
     {
          sum2_red=vis2->red;      sum2_blue=vis2->blue;
sum2_green=vis2->green;
          sum2_red+=vis3->red;  sum2_blue+=vis3->blue;
sum2_green+=vis3->green;
          sum5_red=sum2_red+vis4->red;
          sum5_green=sum2_green+vis4->green;
          sum5_blue=sum2_blue+vis4->blue;
          sum2_red+=vis1->red;  sum2_blue+=vis1->blue;
sum2_green+=vis1->green;
          vis1++;vis2++;vis3++;vis4++;
          dst[first].red=((sum0_red+sum1_red+sum2_red)/9);
          dst[first].blue=((sum0_blue+sum1_blue+sum2_blue)/9);
          dst[first].green=((sum0_green+sum1_green+sum2_green)/9);
          first++;
          dst[next].red=((sum3_red+sum4_red+sum5_red)/9);
```

```
                dst[next].blue=((sum3_blue+sum4_blue+sum5_blue)/9);
                dst[next].green=((sum3_green+sum4_green+sum5_green)/9);
                next++;
               sum0_red=sum1_red;        sum1_red=sum2_red;
                sum0_green=sum1_green;    sum1_green=sum2_green;
                sum0_blue=sum1_blue;  sum1_blue=sum2_blue;
                sum3_red=sum4_red;        sum4_red=sum5_red;
                sum3_green=sum4_green;    sum4_green=sum5_green;
                sum3_blue=sum4_blue;  sum4_blue=sum5_blue;
            }
            first+=dim+2;//next row
            next+=dim+2;//third row
            vis1+=dim;
            vis2+=dim;
            vis3+=dim;
            vis4+=dim;//src[+1][0]
        }
    }
```