**实验报告**

专业：计算机科学与技术

姓名：薛柔

学号：3220104854

日期：2023/10/29

课程名称： 图像信息处理 指导老师： 宋明黎 成绩：

实验名称： 图像的简单几何变换

**一、实验目的和要求**

1. 学会对图像进行简单几何变换操作：平移 (Translation) 、旋转 (Rotation)、缩放 (Scale)、剪切 (Shear)、镜像 (Mirror)。

2. 学会用插值方法填补变换后的空洞

**二、实验内容和原理**

实验内容：

1. 图像的平移 (Translation) 变换

2. 图像的旋转 (Rotation) 变换

3. 图像的缩放 (Scale) 变换

4. 图像的剪切 (Shear) 变换

5. 图像的镜像 (Mirror) 变换

实验原理：

1. 图像的几何变换

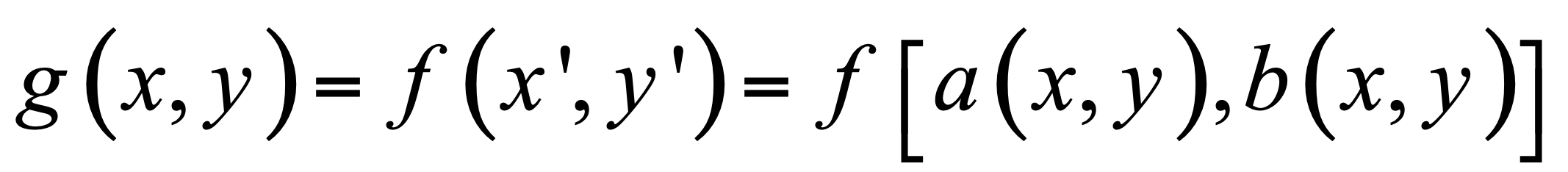
定义： 改变像素的位置来生成新图像。不更改像素值，但更改像素位置。

两种类型：

1）简单变换——变换过程（变换前后的像素位置）是已知的，如平移、镜像、转置、旋转、缩放、剪切等。

2）一般转换——转换过程复杂。很难测量转换参数。通常，当人们修复图像失真时，将采用一般变换。

图像的几何变换可以看成像素坐标的映射，数学形式如下：



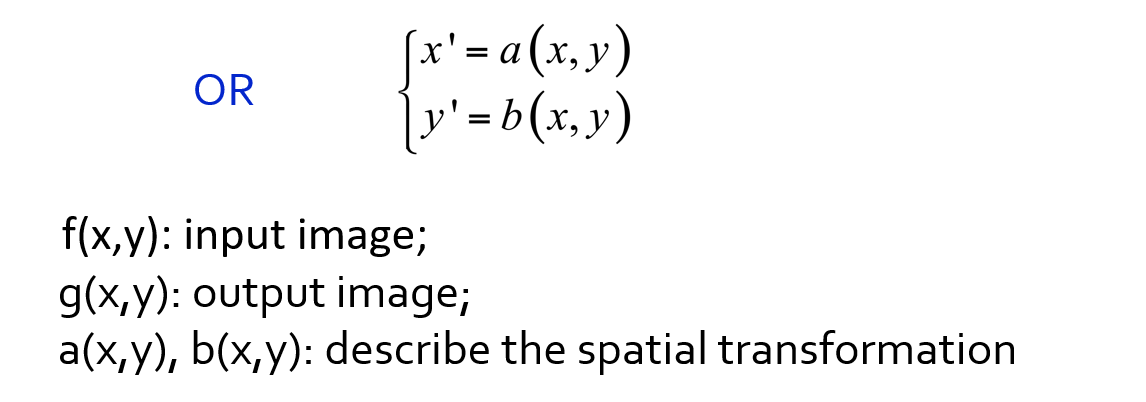


图2.1 几何变换的坐标映射形式

下面是一些一般的几何变换的例子。



图2.2几何变换的例子

下面介绍五种简单的几何变换。

1.1 平移（Translate）

垂直和水平平移图像中的所有像素以生成新图像。方程和案例如下：

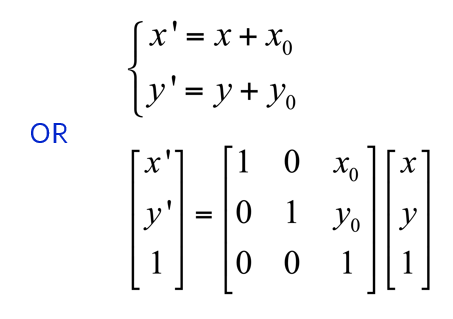


图2.3 平移的方程

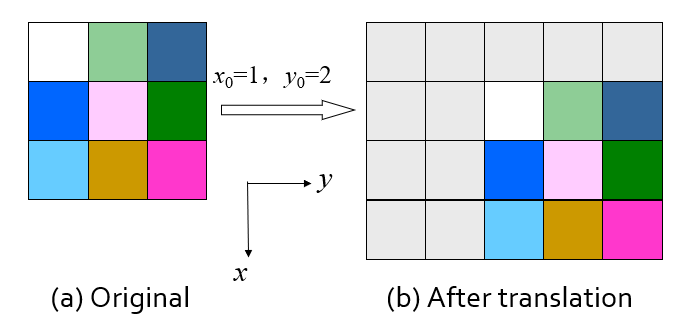


图2.3 平移的例子

注意：平移后的图像与平移前的图像相同，但“画布”应放大，否则会丢失视觉信息。

1.2 镜像（Mirror）

将原始图像绕 X 轴或 Y 轴翻转，并生成与原始图像对称的新图像。方程和案例如下：

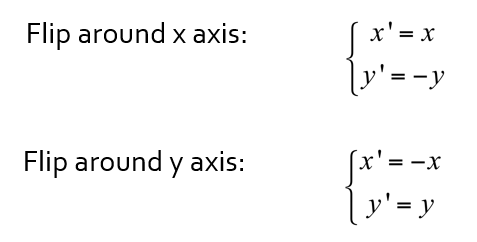


图2.4 镜像的方程

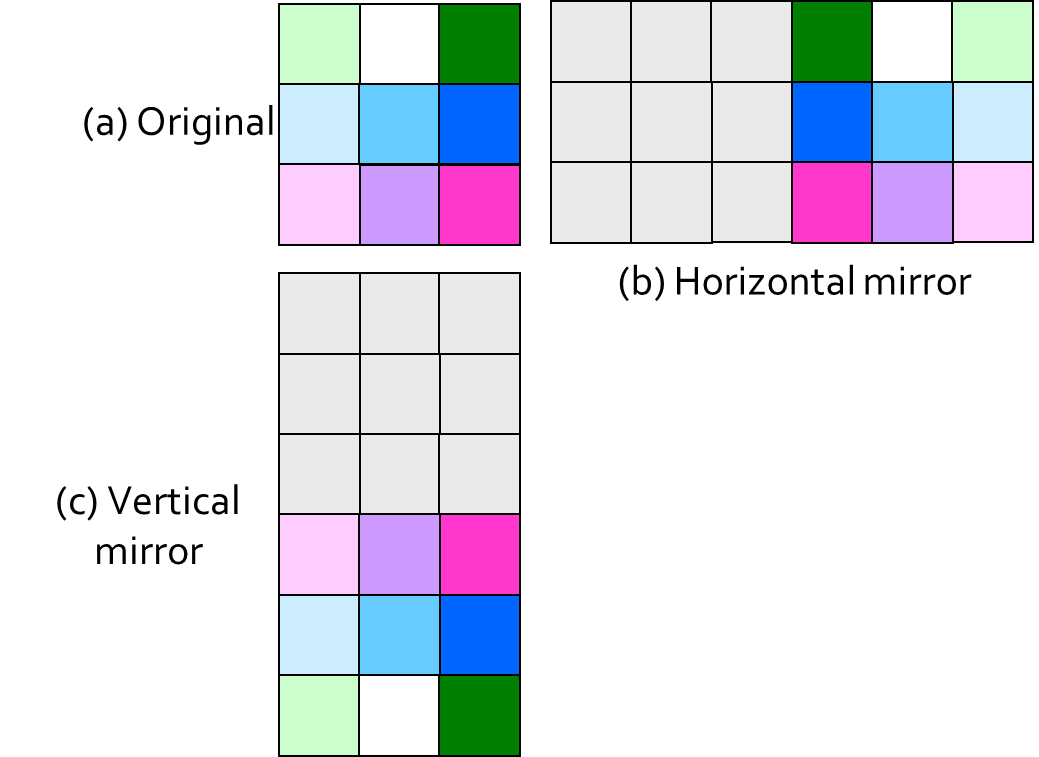


图2.5 镜像的例子

1.3 旋转（Rotation）

围绕原点旋转theta度，生成一个新图像。方程和案例如下：

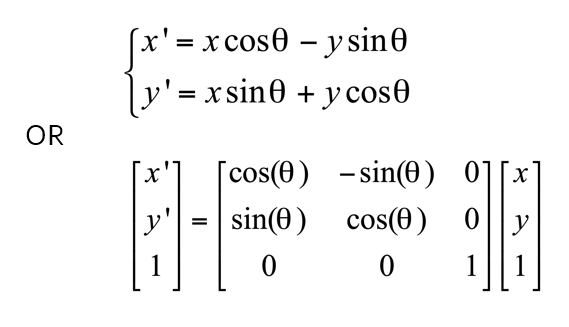


图2.6 旋转的方程

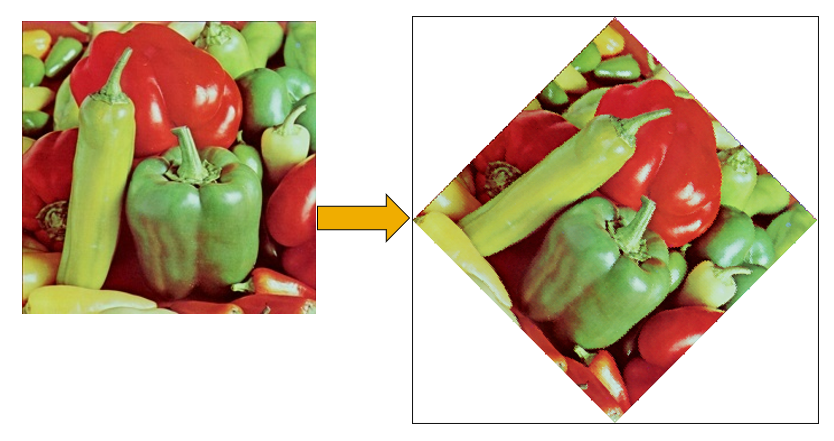


图2.7 旋转的例子

1.4 缩放（Scale）

将图像与系数相乘以生成新图像。

用系数c水平缩放图像（c>1时放大，0<c<1时缩小）;用系数d垂直缩放图像（d>1时放大，0<d<1时缩小）。如果 c=d，则以相同的比例缩放图像。如果不是，则图像将以不同的比例水平和垂直缩放。

缩小（下采样）——通过从原始图像中按预定义的间隔选择像素来构建新图像。放大（简单的上采样）——会有空白的行和列。如果使用插值填充它们，您会在结果中找到一些“马赛克”。

方程和例子如下：

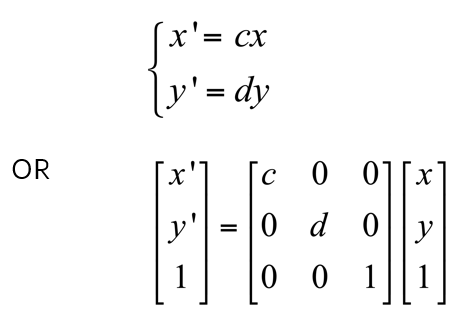
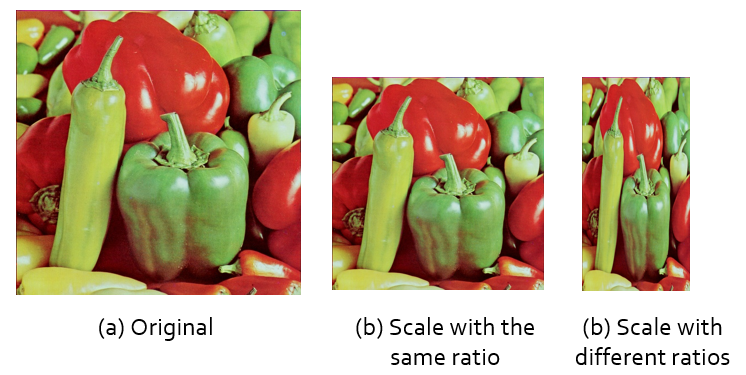


图2.8 缩放的例子



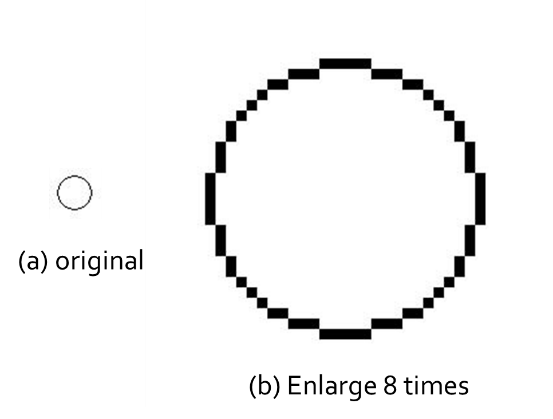


图2.9 缩放的例子

1.5 剪切（Shear）

它是场景在平面上的非垂直投影效果。方程和例子如下：

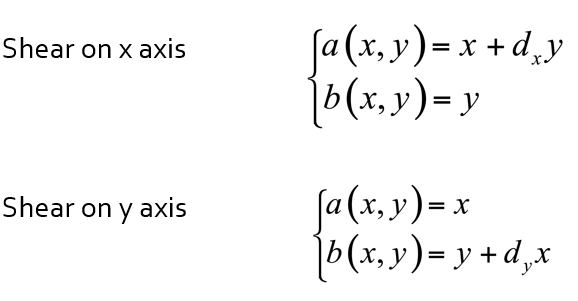


图2.10 剪切的方程

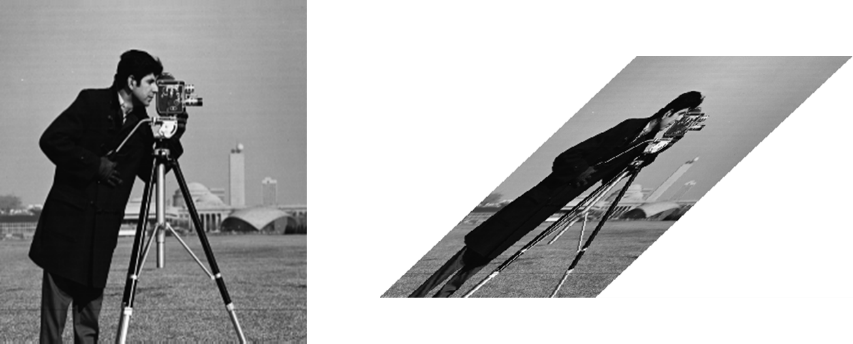


图2.11 x方向上剪切

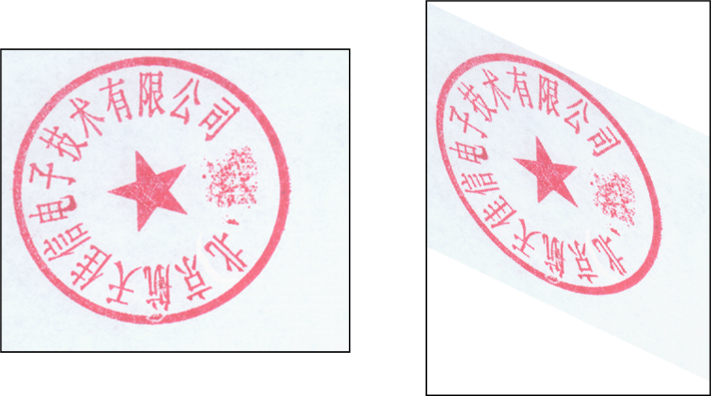


图2.12 y方向上剪切

将上述变换结合起来，即可得到考虑所有简单的几何变换的变换矩阵。

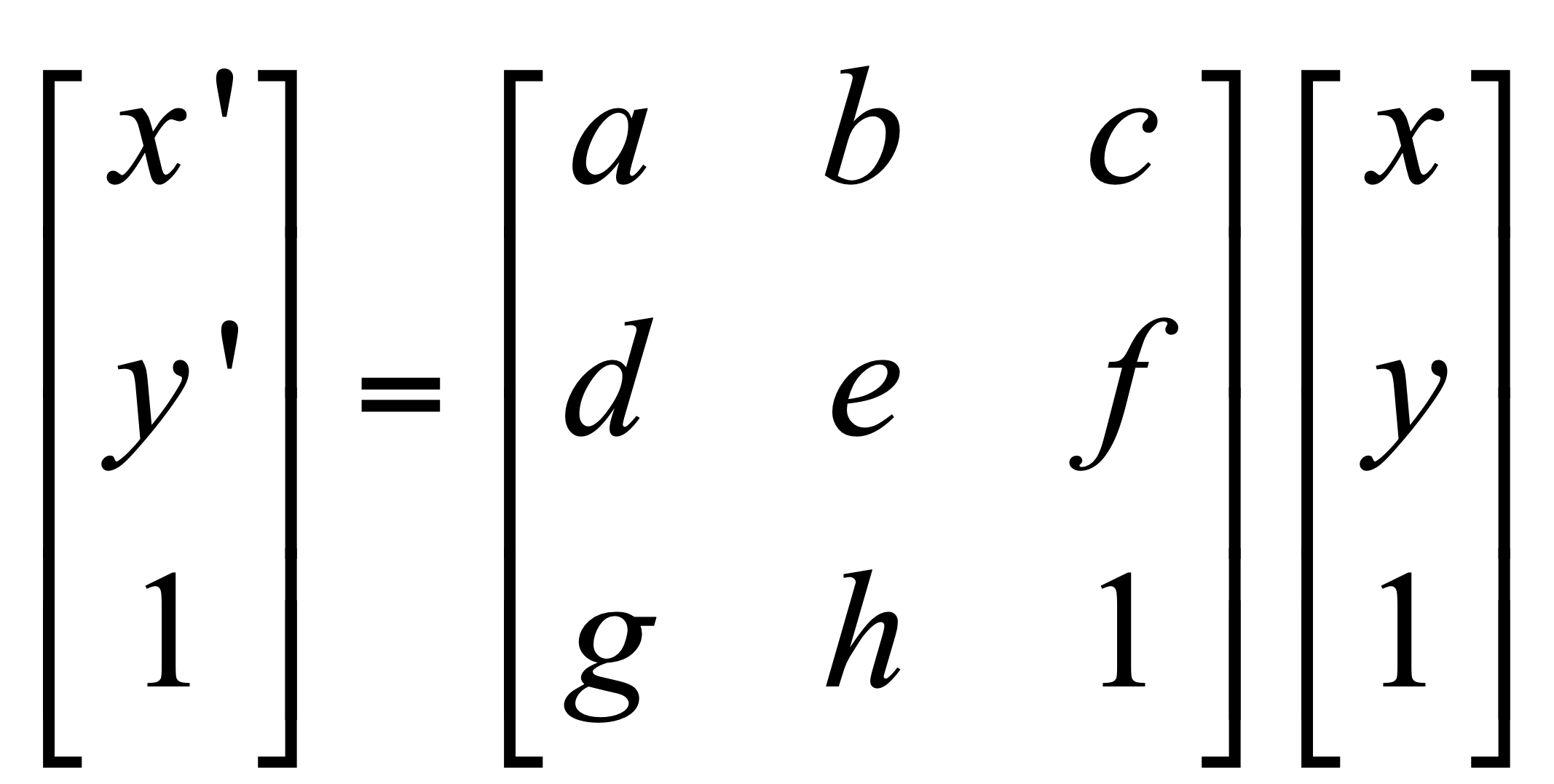


图2.13 变换矩阵

2. 插值

插值是几何变换的典型工具。插值的像素值可以通过使用不同的插值方法获得。下面介绍三种插值方式。

2.1 最邻近插值

输出像素值等于其最近邻的灰度强度。

计算过程：为了计算几何变换后新图像中某一点*P’*处的像素值，可以首先计算该几何变换的逆变换，计算出*P*’所对应的原图像中的位置*P*。通常情况下，*P*的位置不可能正好处在原图像的某一个像素位置上（即*P*点的坐标通常都不会正好是整数）。寻找与*P*点最接近的像素*Q*，把*Q*点的像素值作为新图像中*P*’点的像素值。

优点是简单、易实现。缺点是当图像由明显的几何结构组成时，结果无法保持一致和平滑。

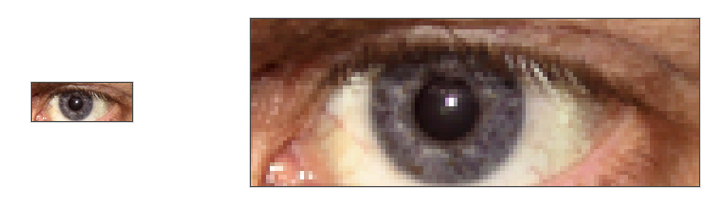


图2.14 最邻近插值的局限

2.2 双线性插值

一维的线性插值的图像和方程如下，即写出直线的方程，将插值点的x值代入，即可得到所求的g（x）值：

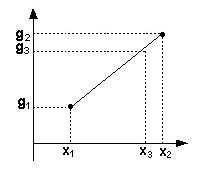
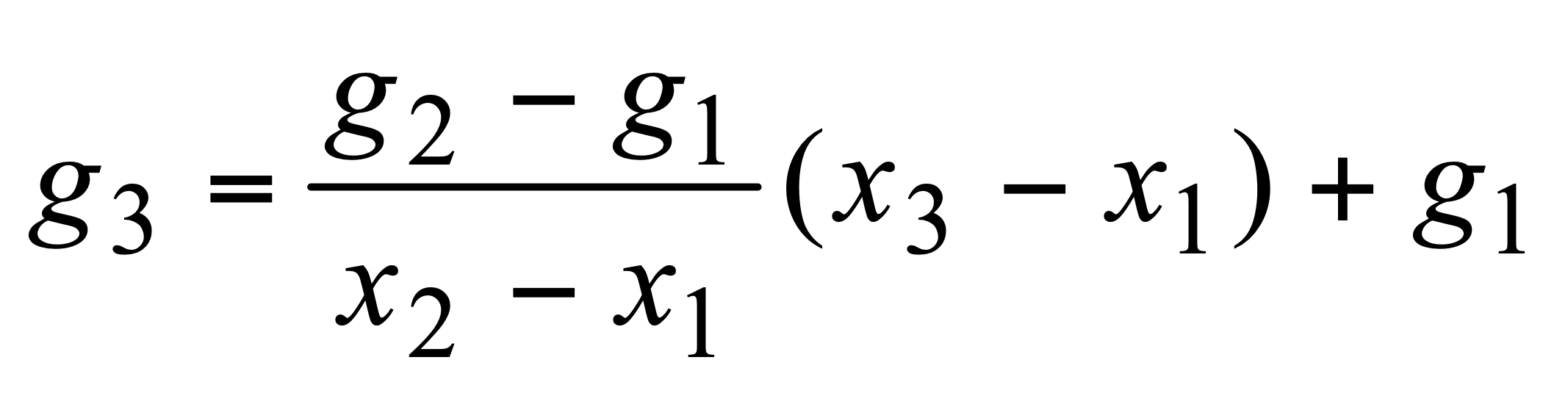


图2.15 一维线性插值

拓展到二维，即根据周围四个点算出平面方程，再将插值点的坐标带入。图示和方程如下：

g ( x , y ) = ax + by + cxy + d，解方程来得到a、b、c、d。

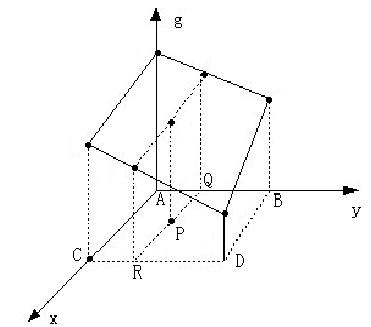


图2.16 双线性插值

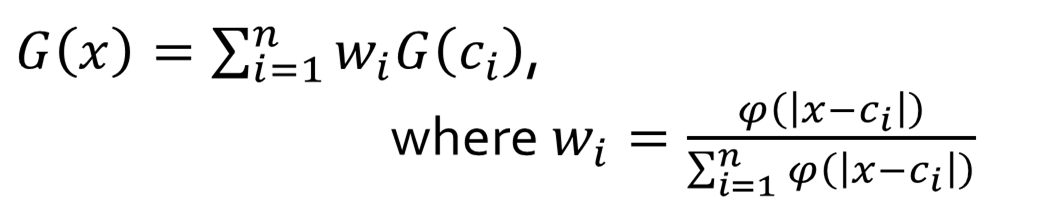
应用在图像处理上的效果如下，可以看出，相较于最邻近插值（下）的明显像素块，双向性插值（上）的结果的过渡更加平滑。



图2.17 双线性插值的效果（上）

2.3 基于径向基函数（RBF）的插值

基于径向基函数（RBF）的插值根据当前点距离所求点的距离赋予不同的权重。离得近的权重大，以得到平滑的图像插值效果。公式以及可选用的基函数如下，其中x可以是多维向量，因此该方法可以用在1D、2D以及更高维的空间上进行插值计算。



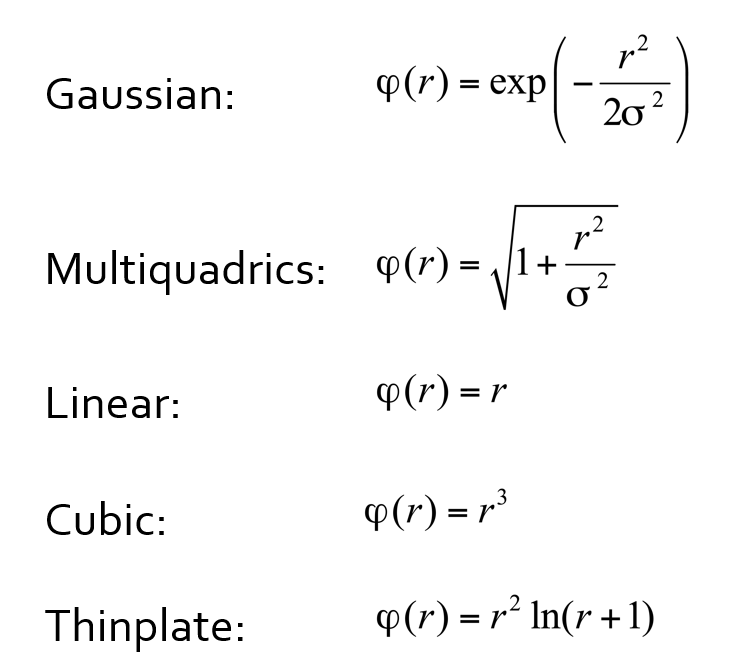


图2.18 RBF插值的公式

**三、实验步骤与分析**

1. 平移操作

先用实验1的方式读取到图像数据，这里不多做赘述。下面是进行平移操作的函数。a是存储原图信息的数组指针，函数实现将图像a向右移动x个像素，向下移动y个像素。实现过程中，是将每个对应位置的像素向右移动x个像素，向下移动y个像素的长度，并将空白部分填充灰色。

其中，输出图像的操作在后面的操作代码中省略。

1. **void** translate(PIXEL \*\*a,**int** x,**int** y)
2. {
3. //translate:平移
4. //a是存储原图信息的数组指针
5. //向右移动x个像素，向下移动y个像素
6. //新的信息头
7. BITMAPINFOHEADER bmih\_tran=bmih;
8. //长宽分别加上位移的像素数
9. bmih\_tran.biWidth=bmih.biWidth+x;
10. bmih\_tran.biHeight=bmih.biHeight+y;
11. //申请空间存储新的图像的数据
12. PIXEL \*new\_pixel[bmih\_tran.biHeight];
13. **for** (**int** i=0;i<bmih\_tran.biHeight;i++){
14. new\_pixel[i] = (PIXEL\*)malloc(**sizeof**(PIXEL)\*(bmih\_tran.biWidth));
15. }
17. **for**(**int** i=0;i<bmih\_tran.biHeight;i++){
18. **for**(**int** j=0;j<bmih\_tran.biWidth;j++){
19. **if**(i-y<0||j-x<0){
20. //空白部分为灰色，在白色和黑色背景下都能看清
21. new\_pixel[i][j].blue=new\_pixel[i][j].green=new\_pixel[i][j].red=200;
22. }**else**{
23. //对应位置的像素平移过去
24. new\_pixel[i][j].blue=a[i-y][j-x].blue;
25. new\_pixel[i][j].green=a[i-y][j-x].green;
26. new\_pixel[i][j].red=a[i-y][j-x].red;
27. }
28. }
29. }
30. //将图输出
31. //打开一个bmp空白图像并写入数据
32. **FILE** \*fp;
33. fp = fopen("translate.bmp", "wb");
34. // fwrite(数据块首地址,元素大小, 元素个数,文件指针)
35. //与原来的图相比，文件头相同
36. fwrite(&bmfh, 14, 1, fp);//文件头
37. fwrite(&bmih\_tran, 40, 1, fp);//信息头
38. //写入位图数据
39. **for**(**int** i=bmih\_tran.biHeight-1;i>=0;i--){
40. fwrite(new\_pixel[i], 3, bmih\_tran.biWidth, fp);
41. }
42. fclose(fp);
43. //释放内存
44. **for** (**int** i=0;i<bmih\_tran.biHeight;i++){
45. free(new\_pixel[i]);
46. }
47. }

2. 旋转操作

运用原理中提到的公式，对图像进行旋转变换，本函数输入的参数theta是逆时针旋转的弧度（而非角度）。比平移复杂一些的是，新的图像的长宽需要一定的计算，并进行四字节的对齐。函数的第一部分计算图像长宽如下：

1. **void** rotation(PIXEL \*\*a,**double** theta)
2. {
3. //Rotation:逆时针旋转theta弧度
4. BITMAPINFOHEADER bmih\_rot=bmih;
5. **FILE** \*fp;
6. **double** min\_x=10000;
7. **double** max\_x=0;
8. **double** max\_y=0;
9. **double** min\_y=10000;
10. //计算图像长宽
11. **for**(**int** i=0;i<bmih.biHeight;i++){
12. **for**(**int** j=0;j<bmih.biWidth;j++){
13. //旋转后的坐标
14. **double** x=j\*cos(theta)+i\*sin(theta);
15. **double** y=j\*sin(theta)-i\*cos(theta);
16. **if**(x>max\_x){
17. max\_x=x;
18. }
19. **if**(x<min\_x){
20. min\_x=x;
21. }
22. **if**(y>max\_y){
23. max\_y=y;
24. }
25. **if**(y<min\_y){
26. min\_y=y;
27. }
28. }
29. }
30. //对齐，每一行必须是四字节的倍数
31. **if**((**int**)(max\_x-min\_x)%4==0){
32. bmih\_rot.biWidth=max\_x-min\_x;
33. }**else**{
34. bmih\_rot.biWidth=((**int**)(max\_x-min\_x)+4)/4\*4;
35. }
36. **if**((**int**)(max\_y-min\_y)%4==0){
37. bmih\_rot.biHeight=max\_y-min\_y;
38. }**else**{
39. bmih\_rot.biHeight=((**int**)(max\_y-min\_y)+4)/4\*4;
40. }

旋转的第二部分的代码是将原图像对应位置的像素用公式旋转过去，这步完成后，会发现新的图像存在很多空洞。这张中间图像会在后面进行展示。

1. //赋初值
2. **for**(**int** i=0;i<bmih\_rot.biHeight;i++){
3. **for**(**int** j=1;j<bmih\_rot.biWidth;j++){
5. new\_pixel[i][j].red=200;
6. new\_pixel[i][j].blue=200;
7. new\_pixel[i][j].green=200;
9. }
10. }
11. //遍历赋值
12. **for**(**int** i=0;i<bmih.biHeight;i++){
13. **for**(**int** j=0;j<bmih.biWidth;j++){
14. **double** x=j\*cos(theta)+i\*sin(theta);
15. **double** y=j\*sin(theta)-i\*cos(theta);
16. **int** new\_i=max\_y-y;
17. **int** new\_j=x-min\_x;
18. **if**(new\_i>=bmih\_rot.biHeight){
19. new\_i=bmih\_rot.biHeight-1;
20. }
21. **if**(new\_i<0){
22. new\_i=0;
23. }
24. **if**(new\_j>=bmih\_rot.biWidth){
25. new\_j=bmih\_rot.biWidth;
26. }
27. **if**(new\_j<0){
28. new\_j=0;
29. }
30. //对应位置的像素旋转过去
31. new\_pixel[new\_i][new\_j].blue=a[i][j].blue;
32. new\_pixel[new\_i][new\_j].green=a[i][j].green;
33. new\_pixel[new\_i][new\_j].red=a[i][j].red;
35. }
36. }

得到了存在空洞的图像后，我们可以利用行插值（用每一行的前一个像素值填补）的方式填补，下面是接下来的行插值实现代码。

1. //遍历，找到每一行的空洞，用前一个元素填充
2. **for**(**int** i=0;i<bmih\_rot.biHeight;i++){
3. **for**(**int** j=1;j<bmih\_rot.biWidth;j++){
4. **if**(new\_pixel[i][j].blue==200&&new\_pixel[i][j].red==200&&new\_pixel[i][j].green==200){
5. **if**(new\_pixel[i][j-1].blue!=200||new\_pixel[i][j-1].red!=200||new\_pixel[i][j-1].green!=200){
6. **if**(new\_pixel[i][j+1].blue!=200||new\_pixel[i][j+1].red!=200||new\_pixel[i][j+1].green!=200){
7. new\_pixel[i][j].red=new\_pixel[i][j-1].red;
8. new\_pixel[i][j].blue=new\_pixel[i][j-1].blue;
9. new\_pixel[i][j].green=new\_pixel[i][j-1].green;
10. }
11. }
12. }
13. }
14. }

3. 缩放操作

这里提供两个系数，ratio\_h和ratio\_w，分别是长宽的放大倍数（可小于1）。这个函数中用到了与前面不同的插值方法——最邻近插值。算出原来像素的位置（一般不是整数），将最邻近的像素的值赋给它，得到一张放大（缩小）后的图像。

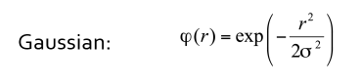
1. **void** scale(PIXEL \*\*a,**double** ratio\_h,**double** ratio\_w)
2. {
3. //Scale：长宽分别放大一定倍数
4. //ratio\_h和ratio\_w：长和宽的放大（缩小）倍数
5. BITMAPINFOHEADER bmih\_sca=bmih;
6. //对齐，略
7. //遍历赋值，使用最近邻插值
8. **for**(**int** i=0;i<bmih\_sca.biHeight;i++){
9. **for**(**int** j=0;j<bmih\_sca.biWidth;j++){
10. **int** old\_j=j/ratio\_w;
11. **int** old\_i=i/ratio\_h;
13. //控制范围
14. **if**(old\_i>=bmih.biHeight){
15. old\_i=bmih.biHeight-1;
16. }
17. **if**(old\_i<0){
18. old\_i=0;
19. }
20. **if**(old\_j>=bmih.biWidth){
21. old\_j=bmih.biWidth;
22. }
23. **if**(old\_j<0){
24. old\_j=0;
25. }
27. new\_pixel[i][j].blue=a[old\_i][old\_j].blue;
28. new\_pixel[i][j].green=a[old\_i][old\_j].green;
29. new\_pixel[i][j].red=a[old\_i][old\_j].red;
31. }
32. }

在后面的探索中，我还在缩放操作中验证了双线性插值和RBF插值的方法。下面的scale\_interpolation函数中使用了双线性插值和RBF插值的方法填补放大后的空洞（由于只是验证插值方法，仅对真色彩图格式的RGB相同的灰度图起效）。为运算简便，双线性插值的方式我是先在x方向上插值得到两个点，再用这两个点对目标像素进行y方向上的插值。

双线性插值的部分如下：

1. **void** scale\_interpolation(PIXEL \*\*a,**double** ratio\_h,**double** ratio\_w,**double** sigma)
2. {
3. //插值法求中间像素
4. BITMAPINFOHEADER bmih\_sca=bmih;
5. //求图像大小、对齐，略
7. //双线性插值
8. **for**(**int** i=0;i<bmih\_sca.biHeight;i++){
9. **for**(**int** j=0;j<bmih\_sca.biWidth;j++){
10. **double** old\_j=j/ratio\_w;
11. **double** old\_i=i/ratio\_h;
12. //取最邻近的四个像素
13. **int** i0=floor(old\_i);
14. **int** j0=floor(old\_j);
15. **int** i1=ceil(old\_i);
16. **int** j1=ceil(old\_j);
17. //控制范围
18. **if**(i1>=bmih.biHeight){
19. i1=bmih.biHeight-1;
20. }
21. **if**(i0<0){
22. i0=0;
23. }
24. **if**(j1>=bmih.biWidth){
25. j1=bmih.biWidth-1;
26. }
27. **if**(j0<0){
28. j0=0;
29. }
31. //在x轴方向上，对上边和下边的R1和R2两个点进行插值
32. **double** temp\_red0=(old\_j-j0)\*(a[i0][j1].red-a[i0][j0].red)+a[i0][j0].red;
33. **double** temp\_red1=(old\_j-j0)\*(a[i1][j1].red-a[i1][j0].red)+a[i1][j0].red;
34. //再根据R1和R2两个点对P插值
35. **int** temp\_red=(old\_i-i0)\*(temp\_red1-temp\_red0)+temp\_red0;
36. new\_pixel[i][j].blue=new\_pixel[i][j].green=new\_pixel[i][j].red=temp\_red;
37. }
38. }

RBF插值部分的代码如下：这里选取



为基函数，对全图的像素算出权重，将加权平均赋给目标像素。函数头的sigma即为公式中的参数，仅在RBF插值中对结果起作用。需要注意的是，由于选取了全图的像素，这个方法运算量非常大，需要消耗大量时间。因此在.exe生成前被注释掉。如果需要调用该方法，需要去掉注释，重新编译。

1. //RBF插值，基函数：exp(-r^2/sigma);
2. **double** sum=0;
3. **double**\* phi[bmih\_sca.biHeight];
4. **for** (**int** i=0;i<bmih\_sca.biHeight;i++){
5. phi[i]=(**double**\*)malloc(**sizeof**(**double**)\*bmih\_sca.biWidth);
6. }
7. **for**(**int** m=0;m<bmih\_sca.biHeight;m++){
8. **for**(**int** n=0;n<bmih\_sca.biWidth;n++){
9. **double** old\_m=m/ratio\_w;
10. **double** old\_n=n/ratio\_h;
11. //算出所有像素的插值系数
12. **for**(**int** i=0;i<bmih.biHeight;i++){
13. **for**(**int** j=0;j<bmih.biWidth;j++){
14. phi[i][j]=exp(-((i-old\_m)\*(i-old\_m)+(j-old\_n)\*(j-old\_n))/sigma);
15. sum+=phi[i][j];
16. }
17. }
18. //双精度降低舍入误差
19. **double** temp\_Y=0;
20. **for**(**int** i=0;i<bmih.biHeight;i++){
21. **for**(**int** j=0;j<bmih.biWidth;j++){
22. temp\_Y+=phi[i][j]/sum\*a[i][j].red;
23. }
24. }
25. new\_pixel[m][n].blue=new\_pixel[m][n].green=new\_pixel[m][n].red=temp\_Y;
26. sum=0;
27. }
28. }

4. 剪切操作

与缩放操作类似，根据原理中的方程做变换，使用了最邻近插值。xory参数可以选择在x方向上剪切（xory=0）或在y方向上剪切（xory=其他）。而剪切系数ratio越大，图像斜得越明显。这里只贴出x方向上剪切的代码，y方向上同理，见源文件。

1. **void** shear(PIXEL \*\*a,**double** ratio,**int** xory)
2. {
3. BITMAPINFOHEADER bmih\_she=bmih;
4. **FILE** \*fp;
5. //x方向上剪切
6. **if**(xory==0){
7. //计算图像大小，对齐
8. bmih\_she.biHeight=bmih.biHeight;
9. **if**((**int**)(bmih.biWidth+bmih.biHeight\*ratio)%4==0){
10. bmih\_she.biWidth=bmih.biWidth+bmih.biHeight\*ratio;
11. }**else**{
12. bmih\_she.biWidth=((**int**)(bmih.biWidth+bmih.biHeight\*ratio)+4)/4\*4;
13. }
15. PIXEL \*new\_pixel[bmih\_she.biHeight];
16. **for** (**int** i=0;i<bmih\_she.biHeight;i++){
17. new\_pixel[i] = (PIXEL\*)malloc(**sizeof**(PIXEL)\*(bmih\_she.biWidth));
18. }
20. //遍历赋值，使用最近邻插值
21. **for**(**int** i=0;i<bmih\_she.biHeight;i++){
22. **for**(**int** j=0;j<bmih\_she.biWidth;j++){
23. **int** old\_j=j-ratio\*i;
24. **int** old\_i=i;
26. //控制范围
27. **if**(old\_i>=bmih.biHeight||old\_i<0||old\_j>=bmih.biWidth||old\_j<0){
28. new\_pixel[i][j].blue=200;
29. new\_pixel[i][j].green=200;
30. new\_pixel[i][j].red=200;
31. }**else**{
32. new\_pixel[i][j].blue=a[old\_i][old\_j].blue;
33. new\_pixel[i][j].green=a[old\_i][old\_j].green;
34. new\_pixel[i][j].red=a[old\_i][old\_j].red;
36. }
37. }
38. }

5. 镜像操作

最后一个图像操作，xory的参数作用同上。不同的是，这个函数不需要插值方法。

1. **void** mirror(PIXEL \*\*a,**int** xory)
2. {
3. //Mirror:x、y方向上镜像图像
4. //x方向上mirror
5. BITMAPINFOHEADER bmih\_mir=bmih;
6. **FILE** \*fp;
7. **if**(xory==0){
8. bmih\_mir.biWidth=bmih.biWidth\*2;
10. PIXEL \*new\_pixel[bmih\_mir.biHeight];
11. **for** (**int** i=0;i<bmih\_mir.biHeight;i++){
12. new\_pixel[i] = (PIXEL\*)malloc(**sizeof**(PIXEL)\*(bmih\_mir.biWidth));
13. }
14. //遍历赋值
15. **for**(**int** i=0;i<bmih\_mir.biHeight;i++){
16. **for**(**int** j=0;j<bmih\_mir.biWidth;j++){
17. **if**(j<bmih.biWidth){
18. //空白部分为灰色，在白色和黑色背景下都能看清
19. new\_pixel[i][j].blue=new\_pixel[i][j].green=new\_pixel[i][j].red=200;
20. }**else**{
21. //对应位置的像素镜像过去
22. new\_pixel[i][j].blue=a[i][bmih\_mir.biWidth-1-j].blue;
23. new\_pixel[i][j].green=a[i][bmih\_mir.biWidth-1-j].green;
24. new\_pixel[i][j].red=a[i][bmih\_mir.biWidth-1-j].red;
25. }
26. }
27. }

**四、实验环境及运行方法**

* 实验环境：Windows 11 系统

gcc 10.3.0 (tdm64-1) x86\_64-w64-mingw32

* 运行方法：先将dip\_hw4.exe文件和42.bmp以及gray\_test.bmp（这张图像用于验证插值的方法）放在同一个目录下，运行dip\_hw4.exe文件，等待1s左右会得到8张新的图像。它们的命名分别为对原始图像进行的操作名（例如x方向上剪切的图像会被命名为shear\_x.bmp）。当终端出现两个“Successfully open the image”时说明我们成功打开了两张原始图像，否则会输出“BMP Image Not Found!”。

如果运行不成功，可以将文件夹中的.c源文件重新编译运行。

**五、实验结果展示**

下面先展示五种基本操作的结果：

图5.1为平移的结果图，参数为：向右平移400px，向下平移200px。

后面的结果未特殊说明，输入图像均为左图42.bmp。

图5.1 输入图像42.bmp（左）和平移后图像translater.bmp（右）

下图5.2为旋转的结果图，参数为：旋转1弧度（逆时针）。左边的图像是像素一对一的坐标变换后的中间图像，因此图像中存在很多黑点。右图是通过行插值方法得到的结果图。图像中的孔被前一个像素填充。

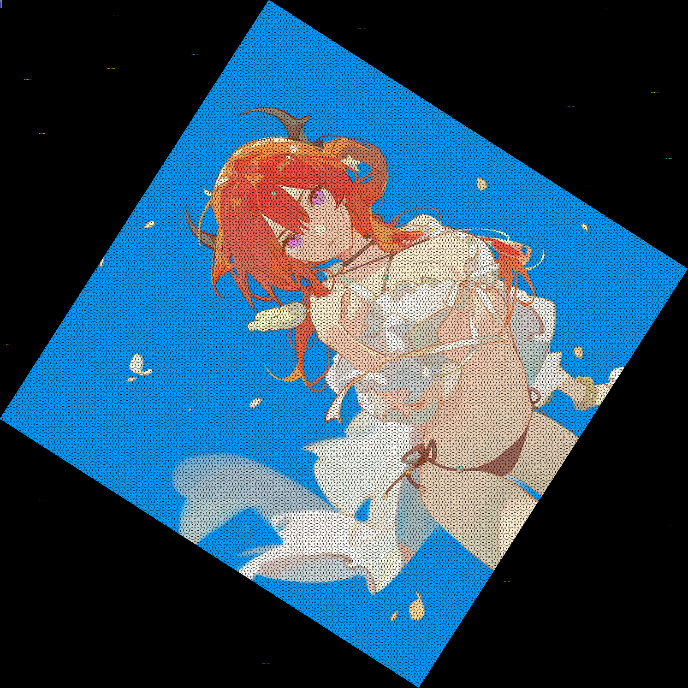
 

图5.2 旋转后有空洞的中间图像rotation\_row.bmp（左）和结果图像rotation.bmp（右）

下图5.3为缩放的结果图，左图参数为：高度变为原来的1.2倍，宽度变为原来的1.8倍。为了看出缩放的效果，特地选用了不同长宽的比例系数。

为验证不同插值方法的效果，我利用实验一中的灰度化方法和scale函数，将原图变为缩小的灰度图gray\_test.bmp（图像太大难以看出不同插值的效果），参数为：高度变为原来的0.2倍，宽度变为原来的0.2倍。

注意，下面的右图为了便于观看经过放大。实际上其与和左图的比例远小于现在的效果。且右图已经跑好，作为测试时的输入。

图5.3 缩放后的图像scale.bmp（左）和gray\_test.bmp（右）

图5.4为x、y方向上剪切的结果图，左图参数为：xory = 1，选择y方向上剪切，剪切系数0.2；右图参数为：xory = 0，选择x方向上剪切，剪切系数0.6。由于原图为正方形，两图之间的对比可以看出，剪切系数越大，剪切程度越高。



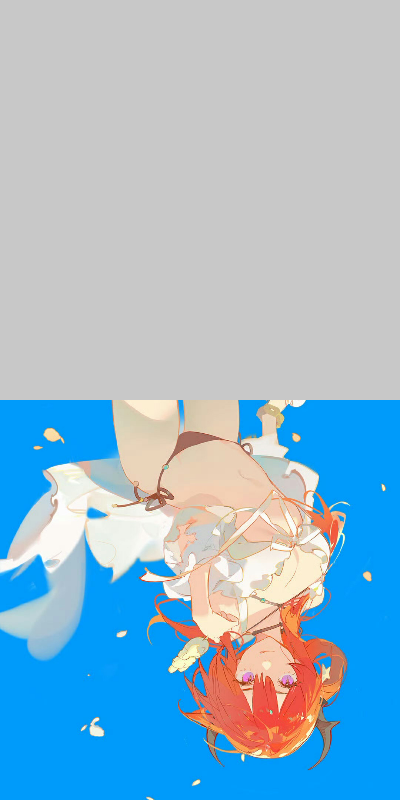
图5.4 剪切后的图像shear\_y.bmp（左）和shear\_x.bmp（右）

图5.5为x、y方向上镜像的结果图，左图参数为：xory = 0，选择x方向上镜像；右图参数为：xory = 1，选择y方向上镜像。

图5.5 镜像后的图像mirror\_x.bmp（左）和mirror\_y.bmp（右）

下面的四张图是实现的不同插值方法的效果对比，参数均为将右边的缩小图长宽均变为原来的三倍。5.6的两张图，左边是最邻近插值的结果，可以在图中看到很多小方块。这是这种方法的局限所在。而右边是双线性插值的结果，图像的过渡相较而言平滑一些。

图5.6 最邻近插值的放大结果scale\_cmp.bmp（左）和线性插值的结果scale\_linear.bmp（右）

下面的两张图是不同的参数的RBF插值的结果。左边参数sigma为1，右边为0.1。可以看出，左边的结果非常平滑，几乎没有小方块，右边的结果与最邻近插值的结果十分相近。得知如果想得到较平滑的结果，应选用较大的sigma。

该方法需要好几分钟的运行时间，因此未编译进程序中。

图5.7 RBF插值方法的结果scale\_RBF.bmp（左）和scale\_RBF\_new.bmp（右）

上述的参数（旋转角度、放大倍数等）均可在源文件中更改来得到其他的图像。

**六、心得体会**

此次实验与前面不同的是，涉及到像素位置的改变。因此，我改变了之前直接按顺序存储像素值的做法，改用二维数组存储特定位置的像素，a[i][j]即为第i行第j列的像素的值。这需要更改很多我的一贯以来的代码架构，是一个难点。这样做以后，像素位置的改变就变得较为简单。

实验还涉及图像大小的改变。我发现，位图信息头中的biWidth和biHeight是改变图像大小的关键变量。图像文件头中的文件大小可以不做改动，对新图像的显示没有作用。乱改文件头和信息头容易造成bmp文件损坏，这是实验中遇到的一个主要的困难。

本次实验，本来想只用最简单的最邻近插值完成，后来发现，双线性插值和RBF插值的代码实现并不困难，便将它们都实现了。不足之处是，RBF的径向距离选取我并未严格考虑，只是简单地将全部图像作为选取的范围，这造成了较大的计算量。

可以看出，复杂度而言，最邻近插值<线性插值<RBF插值，效果上，RBF插值>线性插值>最邻近插值。因此，平衡计算量和效果一直是计算机领域需要考虑的事情之一。