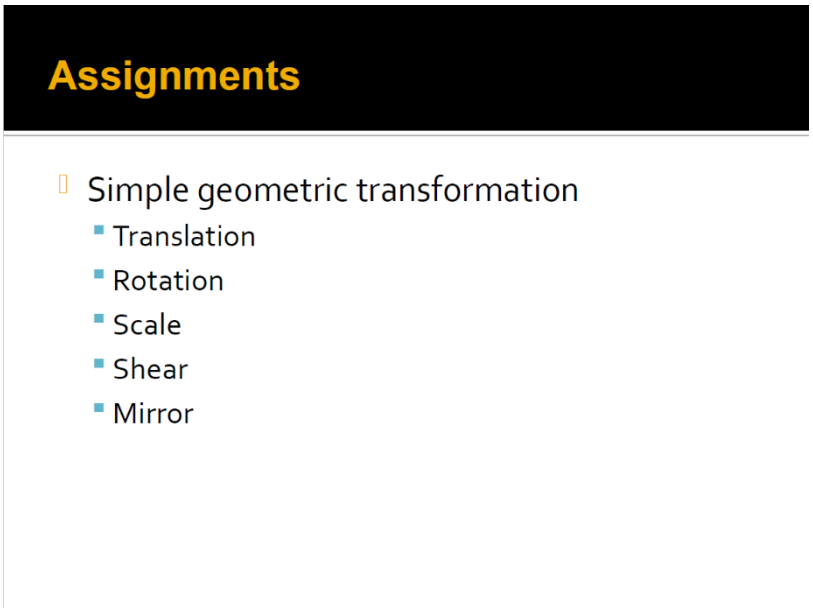
浙江大学实验报告

课程名称：图像信息处理 指导老师：宋明黎 成绩：

实验名称：图像几何处理

一、实验目的和要求



二、实验内容和原理

1、图像平移

Simple geometric transformation

Translation

Translate all the pixels in an image vertically and horizontally to produce a new image.

Simple geometric transformation

Translation——Equation

$$\begin{cases} x' = x + x_0 \\ y' = y + y_0 \end{cases}$$

OR

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Each pixel in the original image is translated x_0 and y_0 respectively.

2、图像镜面翻转

Simple geometric transformation

Mirror——Concept

flip around x axis or y axis, and produce a new image symmetric to the original one.

Simple geometric transformation

Mirror——Equation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

When $S_x = 1$， and $S_y = -1$, flip around the x axis
When $S_x = -1$， and $S_y = 1$, flip around the y axis

3、图像旋转

Simple geometric transformation

Rotation——Concept

Rotate the image around the origin θ, and produce a new image.

Simple geometric transformation

Rotation——Equation

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

OR

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Simple geometric transformation

Rotation——"Hole"problem

Fill by "interpolation"

Row interpolation——find the holes in each row, and fill them with the intensity of the pixel before it in the row.

4、图像缩放

Simple geometric transformation

Scale——Concept

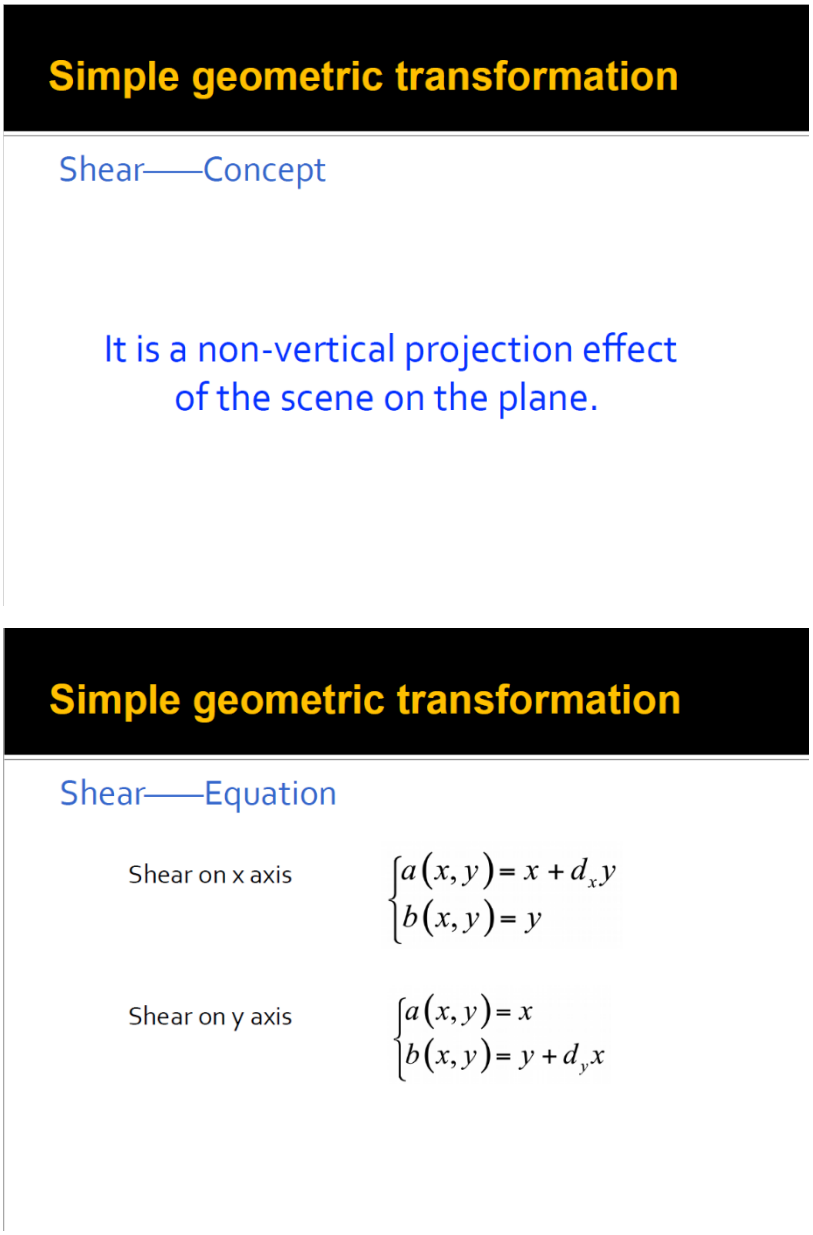
Multiple the image with a coefficient to produce a new image.

Simple geometric transformation

Scale——Equation $\begin{cases} x' = cx \\ y' = dy \end{cases}$

- Scale the image horizontally with coefficient c (enlarge when c>1, shrink when 0<c<1); scale the image vertically with coefficient d (enlarge when d>1, shrink when 0<d<1)
- If c=d, the image is scaled with the same ratio. If NOT, the image is scaled horizontally and vertically with different ratios.
- Shrink (down-sampling)——construct a new image by select pixels by predefined intervals from the original image.
- Enlarge (a simple up-sampling)——there will be blank rows and columns. If you fill them by using interpolation, you will find some "mosaics" in the result.

5、图像剪切



三、实验步骤与分析

1、读取 BMP 文件的基本信息

沿用上次作业的 ReadBmp 函数

2、图像平移

对每个像素点进行移位，移动后无像素点的位置置为黑色画布。注意 bmp 图像原点从左下角开始，在 y 轴处理上要和正常的左上角存储的像素点数组相反，即第一行对应最后一行。

```
void translation(int x, int y)
{
    int moveX, moveY, saveX, saveY;
    //对4取整
    if (x % 4 == 0)
        moveX = x;
    else
        moveX = x / 4 * 4;
    moveY = y;
    saveX = moveX; saveY = moveY;
    if (moveX < 0)
        moveX = -moveX;
    if (moveY < 0)
        moveY = -moveY;
    //新坐标
    int nH = pH + moveY;
    int nW = pW + moveX;
    moveX = saveX; moveY = saveY;
    //新像素数
    int newSize = nH * nW;
    P *Q = (P*)malloc(sizeof(P) * newSize);
    memset(Q, 0, sizeof(P) * newSize);
    //处理每个像素
    if (moveY < 0)
    {
        moveY = -moveY;
        for (int i = 0; i < nH; i++)
        {
            for (int j = 0; j < nW; j++)
            {
                if (nH - i <= moveY)
                {
                    Q[i * nW + j].red = Q[i * nW + j].blue = Q[i * nW + j].green = 0;
                    continue;
                }
                if (j < moveX || nW - j <= -moveX)
                    Q[i * nW + j].red = Q[i * nW + j].blue = Q[i * nW + j].green = 0;
                else if (moveX >= 0)
                    Q[i * nW + j] = imgp[i * pW + (j - moveX)];
                else if (moveX < 0)
                    Q[i * nW + j] = imgp[i * pW + j];
            }
        }
    }
    else
    {
        for (int i = 0; i < nH; i++)
        {
            for (int j = 0; j < nW; j++)
            {
                if (i < moveY)
                {
                    Q[i * nW + j].red = Q[i * nW + j].blue = Q[i * nW + j].green = 0;
                    continue;
                }
                if (j < moveX || nW - j <= -moveX)
                    Q[i * nW + j].red = Q[i * nW + j].blue = Q[i * nW + j].green = 0;
                else if (moveX >= 0)
                    Q[i * nW + j] = imgp[(i - moveY) * pW + (j - moveX)];
                else if (moveX < 0)
                    Q[i * nW + j] = imgp[(i - moveY) * pW + j];
            }
        }
    }

    //建立新的文件
    FILE* fpo1;
    fpo1 = fopen("trans.bmp", "wb");
    //新的文件头信息头
    BITMAPFILEHEADER nHead = fileHeader;
    BITMAPINFOHEADER newInfo = infoHeader;
    //新的图像大小
    nHead.bfSize = (DWORD)(fileHeader.bfSize - size * 3 + newSize * 3);
    //新的图像高度和宽度
    newInfo.biHeight = (DWORD)nH;
    newInfo.biWidth = (DWORD)nW;
    newInfo.biSizeImage = (DWORD)(newSize * 3);
    fwrite(&nHead, 1, sizeof(BITMAPFILEHEADER), fpo1);
    fwrite(&newInfo, 1, sizeof(BITMAPINFOHEADER), fpo1);
    if (QUAD)
        fwrite(QUAD, 1, sizeof(RGBQUAD) * Clr, fpo1);
    fwrite(Q, 1, sizeof(P) * (newSize), fpo1);
    fclose(fpo1);
    //释放空间
    free(Q);
}
```

2、图像镜面翻转

利用数学原理，x=-x，y=-y。sx=-1 关于 y 轴翻转，sy=-1 关于 x 轴翻转

```
void mirror(int sx, int sy) {
    P* Q = (P*)malloc(sizeof(P) * size);
    memset(Q, 0, sizeof(P) * size);
    if (sx == 1 && sy == -1)
        for (int i = 0; i < pH; i++)
            for (int j = 0; j < pW; j++)
                Q[i * pW + j] = imgp[i * pW + pW - 1 - j];  //horizontal mirror
    else if (sx == -1 && sy == 1)
        for (int i = 0; i < pH; i++)
            for (int j = 0; j < pW; j++)
                Q[i * pW + j] = imgp[(pH - i - 1) * pW + j];    //vertical mirror
    else if (sx == -1 && sy == -1)
        for (int i = 0; i < pH; i++)
            for (int j = 0; j < pW; j++)
                Q[i * pW + j] = imgp[(pH - i - 1) * pW + pW - 1 - j];
//horizontal && vertical mirror
    //建新文件
    FILE* fpo1;
    fpo1 = fopen("mirr.bmp", "wb");
    fwrite(&fileHeader, 1, sizeof(BITMAPFILEHEADER), fpo1);
    fwrite(&infoHeader, 1, sizeof(BITMAPINFOHEADER), fpo1);
    fwrite(Q, 1, sizeof(P) * size, fpo1);
    fclose(fpo1);
    free(Q);
}
```

3、图像旋转

以画布中心为原点进行旋转，将扩展后空的画布部分填白。

```
void rotation(double theta)
{
    int nW;
    int nH;
    tXY(&nW, &nH, theta);
    if (nW % 4 != 0)
        nW = (nW / 4 + 1) * 4;
    int newSize = nH * nW;
    P* Q = (P*)malloc(sizeof(P) * newSize);
    memset(Q, 0, sizeof(P) * newSize);
    int* tK = (int*)malloc(sizeof(int) * newSize);
    int* minIndexX = (int*)malloc(sizeof(int) * nH);
    int* maxIndexX = (int*)malloc(sizeof(int) * nH);
    int* minIndexY = (int*)malloc(sizeof(int) * nW);
    int* maxIndexY = (int*)malloc(sizeof(int) * nW);

    for (int i = 0; i < nH; i++)
        minIndexX[i] = 999999;
    for (int i = 0; i < nW; i++)
        minIndexY[i] = 999999;
    memset(maxIndexX, 0, sizeof(int) * nH);
    memset(maxIndexY, 0, sizeof(int) * nW);

    for (int i = 0; i < newSize; i++)
        tK[i] = 1;

    int moveX = (nW - pW) / 2;
    int moveY = (nH - pH) / 2;

    for (int i = 0; i < pH; i++)
    {
        for (int j = 0; j < pW; j++)
        {
            int tmpX = (int)xAfterRot(j, i, theta) + moveX;
            int tmpY = (int)yAfterRot(j, i, theta) + moveY;
            Q[tmpY * nW + tmpX] = imgp[i * pW + jj;
            tK[tmpY * nW + tmpX] = 0;
            minIndexX[tmpY] = min(tmpX, minIndexX[tmpY]);
            maxIndexX[tmpY] = max(tmpX, maxIndexX[tmpY]);
            minIndexY[tmpX] = min(tmpY, minIndexY[tmpX]);
            maxindexY[tmpX] = max(tmpY, maxindexY[tmpX]);
        }
    }

    //first do the horizontal interpolation
    for (int i = 0; i < nH; i++)
    {
        int last = -1;
        for (int j = 0; j < nW; j++)
        {
            if (j < minindexX[i] || j > maxindexX[i])
            {
                Q[i * nW + j].red = 255;
                Q[i * nW + jj.green = 255;
                Q[i * nW + j].blue = 255;
                continue;
            }
            if (tK[i * nW + j] == 0)
            {
                last = j;
                continue;
            }
            else
            {
                int k = j;
                //find next position which has value
                while (k < nW && tK[i * nW + k] == 1)
                    k++;
                if (k >= nW)
                {    //in the last position of the line
                    if (last == -1)
                        break;
                    Q[i * nW + j] = Q[i * nW + last];
                }
                else
                {
                    if (last == -1)     //in the first position of the line
                        Q[i * nW + j] = Q[i * nW + k];
                    else
                    {       //in the middle
                        float coe = (j - last) * 1.0 / (k - last);
                        Q[i * nW + j].red = (BYTE)(((int)Q[i * nW + k].red - (int)Q[i * nW + last].red) * coe + (int)Q[i * nW +
last].red);

                        Q[i * nW + j].blue = (BYTE)(((int)Q[i * nW + k].blue - (int)Q[i * nW + last].blue) * coe + (int)Q[i * nW
 + last].blue);

                        Q[i * nW + j].green = (BYTE)(((int)Q[i * nW + k].green - (int)Q[i * nW + last].green) * coe + (int)Q[i
 * nW + last].green);
                    }
                }
            }
            tK[i * nW + j] = 0;
        }
    }
    //then do the vertical interpolation
    for (int i = 0; i < nW; i++)
    {
        int last = -1;
        for (int j = 0; j < nH; j++)
        {
            if (j < minIndexY[i] || j > maxIndexY[i])
            {

                continue;
            }
            if (tK[j * nW + i] == 0)
            {

                last = j;
                continue;
            }
            else
            {
                int k = j;
                //find next position which has value
                while (k < nH && tK[k * nW + i] == 1)
                {

                    k++;
                }
                if (k >= nH)
                {    //in the last position of the line
                    if (last == -1)
                        break;
                    Q[j * nW + i] = Q[last * nW + i];
                }
                else
                {
                    if (last == -1)
                    {   //in the first position of the line
                        Q[j * nW + i] = Q[k * nW + i];
                    }
                    else
                    {       //in the middle
                        float coe = (j - last) * 1.0 / (k - last);
                        Q[j * nW + i].red = (BYTE)(((int)Q[k * nW + i].red - (int)Q[last * nW + i].red) * coe + (int)Q[last * nW
 + i].red);

                        Q[j * nW + i].blue = (BYTE)(((int)Q[k * nW + i].blue - (int)Q[last * nW + i].blue) * coe + (int)Q[last
 * nW + i].blue);

                        Q[j * nW + i].green = (BYTE)(((int)Q[k * nW + i].green - (int)Q[last * nW + i].green) * coe + (int)Q[
last * nW + i].green);
                    }
                }
            }
            tK[j * nW + i] = 0;
        }
    }

    //计算生成的数据写入文件
    FILE* fpo1;
    fpo1 = fopen("rotat.bmp", "wb");
    BITMAPFILEHEADER nHead = fileHeader;
    BITMAPINFOHEADER newInfo = infoHeader;
    nHead.bfSize = (DWORD)(nHead.bfSize - size * 3 + newSize * 3);
    newInfo.biHeight = (DWORD)nH;
    newInfo.biWidth = (DWORD)nW;
    newInfo.biSizeImage = (DWORD)(newSize * 3);
    fwrite(&nHead, 1, sizeof(BITMAPFILEHEADER), fpo1);
    fwrite(&newInfo, 1, sizeof(BITMAPINFOHEADER), fpo1);
    fwrite(Q, 1, sizeof(P) * (newSize), fpo1);
    fclose(fpo1);
    free(Q);
    free(tK);
    free(minIndexX);
    free(maxIndexX);
    free(minIndexY);
    free(maxIndexY);
}
```

4、图像缩放

首先计算缩放后的画布中各点在原画布中的坐标点，超出全部留白，不然就将原先点填入。对于放大的，在像素间隙内进行填充；对于缩小的，由于缩小时部分整数会变成浮点数，因此对于变成浮点数的像素点全部舍弃，从而达成缩小。

```
void scale(float x, float y)
{
    //新的图像宽高分别为
    int nH = (int)(oH * y);
    int nW = (int)(oW * x);
    int nW;
    //四字节对
    if (nW % 4 == 0)
        nW = nW;
    else
        nW = (nW / 4 + 1) * 4;
    //为新图像开辟空间
    int newSize = nH * nW;
    pR Q = (pR)malloc(sizeof(R) * newSize);
    int * tX = (int*)malloc(sizeof(int) * newSize);
    //为新图像初始化内存
    memset(Q, 0, sizeof(R) * newSize);
    memset(tX, 0, sizeof(int) * newSize);
    //对缩放后图像操作
    for (int i = 0; i < nH; i++)
    {
        for (int j = 0; j < nW; j++)
        {
            float tY, tX;
            int intX, intY;
            //将scale后 i图像对应原图像坐标
            tX = (j * 1.0 / x);
            tY = (i * 1.0 / y);
            intX = (int)tX;
            intY = (int)tY;

            if (tX >= oW || tY >= oH)
                Q[i * nW + j].red = Q[i * nW + j].green = Q[i * nW + j].blue = 255;
            else if (tX - intX > 0 || tY - intY > 0)   //缩放
            {
                Q[i * nW + j].red = Q[i * oW + j].green = Q[i * nW + j].blue = 255;
                tX[i * nW + j] = 1;
            }
            else
                Q[i * nW + j] = bmp[intY * oW + intX];
        }
    }
    //行扫描scale
    for (int i = 0; i < nH; i++)
    {
        int last = -1;
        for (int j = 0; j < nW; j++)
        {
            if (tX[i * nW + j] == 0)
            {
                last = j;
                continue;
            }
            else
            {
                int k = j;
                //找到下一个有值的点
                while (k < nW && tX[i * nW + k] == 1)
                    k++;
                if (k >= nW)
                {   //行尾空
                    if (last == -1)
                        break;
                    Q[i * nW + j] = Q[i * nW + last];
                }
                else
                {
                    if (last == -1) //行头
                        Q[i * nW + j] = Q[i * nW + k];
                    else
                    {
                        float coe = (j - last) * 1.0 / (k - last);
                        Q[i * nW + j].red = (BYTE)(((int)Q[i * nW + k].red - (int)Q[i * nW + last].red) * coe + (int)Q[i * nW +
last].red);

                        Q[i * nW + j].blue = (BYTE)(((int)Q[i * nW + k].blue  - (int)Q[i * nW + last].blue) * coe + (int)Q[i * nW
+ last].blue);

                        Q[i * nW + j].green = (BYTE)(((int)Q[i * nW + k].green - (int)Q[i * nW + last].green) * coe + (int)Q[i
* nW + last].green);
                    }
                }
            }
            tX[i * nW + j] = 0;
        }
    }
    //纵下scale
    for (int i = 0; i < nW; i++)
    {
        int last = -1;
        for (int j = 0; j < nH; j++)
        {
            if (tX[j * nW + i] == 0)
            {
                last = j;
                continue;
            }
            else
            {
                int k = j;
                //find next position which has value
                while (k < nH && tX[k * nW + i] == 1)
                {
                    k++;
                }
                if (k >= nW)
                {   //行尾
                    if (last == -1)
                        break;
                    Q[i * nW + i] = Q[last * nW + i];
                }
                else
                {
                    if (last == -1)
                    {   //行头
                        Q[j * nW + i] = Q[k * nW + i];
                    }
                    else
                    {
                        float coe = (j - last) * 1.0 / (k - last);
                        Q[i * nW + i].red = (BYTE)(((int)Q[k * nW + i].red   (int)Q[last * nW + i].red) * coe + (int)Q[last * nW
+ i].red);

                        Q[i * nW + i].blue = (BYTE)(((int)Q[k * nW + i].blue   (int)Q[last * nW + i].blue) * coe + (int)Q[last
* nW + i].blue);

                        Q[i * nW + i].green = (BYTE)(((int)Q[k * nW + i].green   (int)Q[last * nW + i].green) * coe + (int)Q[
last * nW + i].green);
                    }
                }
            }
            tX[j * nW + i] = 0;
        }
    }
    //写文件
    FILE* fout;
    fout = fopen("scale.bmp", "wb");
    BITMAPFILEHEADER bHead = fileHeader;
    BITMAPINFOHEADER newInfo = infoHeader;
    bHead.bfSize = (DWORD)(bHead.bfSize - size * 3 + newSize * 3);
    newInfo.biHeight = (DWORD)nH;
    newInfo.biWidth = (DWORD)nW;
    newInfo.biSizeImage = (DWORD)(newSize * 3);
    fwrite(&bHead, 1, sizeof(BITMAPFILEHEADER), fout);
    fwrite(&newInfo, 1, sizeof(BITMAPINFOHEADER), fout);
    fwrite(Q, 1, sizeof(R) * newSize, fout);
    fclose(fout);
    free(Q);
    free(tX);
}
```

5、图像剪切

简易版的旋转+缩放。

```c
void shear(double angle)
{
    int nH, nW, newSize;
    P* Q;
    int* tK;

    nW = pW;

    nH = pH + tan(angle) * pW;
    newSize = nW * nH;
    Q = (P*)malloc(sizeof(P) * newSize);
    tK = (int*)malloc(sizeof(int) * newSize);
    //判断所有像素是否已经有信息的标记数组
    for (int i = 0; i < newSize; i++)
        tK[i] = 1;
    //原来的
    for (int i = 0; i < pH; i++)
    {
        for (int j = 0; j < pW; j++)
        {
            //求变换后的横纵坐标，并赋值对应的像素
            int tmpX = j;
            int tmpY = (int)(i + tan(angle) * j);
            Q[tmpY * nW + tmpX] = imgp[i * pW + j];
            //有信息后标记为0
            tK[tmpY * nW + tmpX] = 0;
        }
    }
    //没图像的位置都用白色填充
    for (int i = 0; i < newSize; i++)
        if (tK[i] == 1)
            Q[i].red = Q[i].green = Q[i].blue = 255;
    //新文件
    FILE* fpo1;
    fpo1 = fopen("shear.bmp", "wb");
    BITMAPFILEHEADER nHead = fileHeader;
    BITMAPINFOHEADER newInfo = infoHeader;
    nHead.bfSize = (DWORD)(nHead.bfSize - size * 3 + newSize * 3);
    newInfo.biHeight = (DWORD)nH;
    newInfo.biWidth = (DWORD)nW;
    newInfo.biSizeImage = (DWORD)(newSize * 3);
    fwrite(&nHead, 1, sizeof(BITMAPFILEHEADER), fpo1);
    fwrite(&newInfo, 1, sizeof(BITMAPINFOHEADER), fpo1);
    fwrite(Q, 1, sizeof(P) * (newSize), fpo1);
    fclose(fpo1);
    free(Q);
    free(tK);
}
```

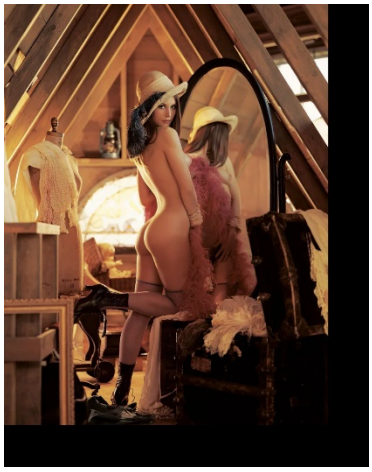
四、实验环境及运行方法

编译环境：C 语言，使用最新 C11 标准。ide 使用 visual studio。用 dev 可直接打开 main.c 源文件进行编译。

测试方法：断点单步测试、修改参数查看输出图片、与 matlab 结果相比较

五、实验结果展示

translation：



(-90,90)

(90,90)

(90,-90)

mirror：

(-1,1)

(-1,-1)

(1,-1)

rotation && shear：

scale：

(1,4)

(0.5,1)

(5,5) 3550 x 4540

六、心得体会

本次作业加深了我对像素点和画布的认识。