



# 图像信息处理实验报告

**Digital Image Processing (Experiments)**

|  |  |
| --- | --- |
| 姓 名 |  |
| 指导老师 |  |
| 学 号 |  |
| 专业班级 |  |

**二〇一九年**

**秋冬学期**

## 实验四

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称：** | 作业4 | **指导老师：** |  | **成绩：** |  |

### 一、实验目的和要求（必填）

实现一些图像的简单几何变化

1、图像的平移（Translation）

2、图像的旋转（Rotation）

3、图像的放缩（Scale）

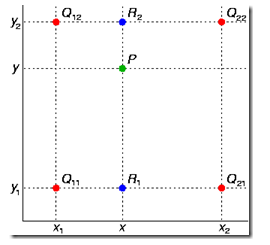
4、图像的错切（Shear）

5、图像的镜像变化（Mirror）

### 二、实验内容和原理（必填）

#### 1、双线性插值法

图像经过某些几何变化后，像素的整数坐标有极大的可能变为小数，但图像的像素点里是没有小数坐标的，因此我们会从输出的图像像素出发，映射到原图的P点，由P点周围最接近的4个像素坐标点加权平均获得P点的颜色值。

如图：

d1=COLOR(Q12)\*DISTANCE(Q22\_R2)+ COLOR(Q22)\*DISTANCE(Q12\_R2)

d2=COLOR(Q11)\*DISTANCE(Q21\_R1)+ COLOR(Q21)\*DISTANCE(Q11\_R1)

COLOR(P)=d1\*DISTANCE(R1\_P)+ d2\*DISTANCE(R2\_P)

#### 2、坐标平移变换公式

#### 3、平面内直角坐标系中坐标旋转变换公式

（x，y）--旋转角度β-->（x1，y1）

顺时针旋转：

**x1=xcos(β)-ysin(β)**

**y1=ycos(β)+xsin(β)**

逆时针旋转：

**x1=xcos(β)+ysin(β);**

**y1=ycos(β)-xsin(β);**

**逆时针旋转变化公式：**

#### 4、图像缩放变换公式

#### 5、图像错切变换公式

x坐标轴上进行错切：

y坐标轴上进行错切：

#### 6、图像镜像变换公式

x轴方向对称：

y轴方向对称：

### 三、源代码与分析

#### 1、图像几何变化主函数

#include <stdio.h>

#include <stdlib.h> //malloc free

#include<math.h>

#include "bmp.h"

//定义画布长宽值

#define BMP\_WIDTH  1000

#define BMP\_HEIGHT 1000

#define BMP\_PERW   3

#define BM 19778

#define MAXSIZE 10000

//输出像素值的rgb值数组

//所以在对坐标[x,y]定点时,应该这样调用 buff[y][x][..]

//第三维：0->R  1->G  2->B

unsigned char buff[BMP\_HEIGHT][BMP\_WIDTH][BMP\_PERW];

//输入像素值的rgb值数组

unsigned char R[MAXSIZE][MAXSIZE], G[MAXSIZE][MAXSIZE], B[MAXSIZE][MAXSIZE];

//指针偏移量

unsigned int OffSet = 0;

//输入图片的长款值

int width, height;

//打开源文件

FILE\* openFile(char name[]);

//原图信息的读取

void readBMP(FILE \*fpbmp);

int main(int argc, char \*argv[])

{

    int x, y;

    FILE\* fpbmp;

    fpbmp = openFile("input.bmp");

    readBMP(fpbmp);

    //---------- 创建一张图片 ----------

    //initial

    for (x = 0; x < BMP\_WIDTH; x++)

    {

        for (y = 0; y < BMP\_HEIGHT; y++)

        {

            buff[y][x][0] = R[y\*height / BMP\_HEIGHT][20];

            buff[y][x][1] = G[y\*height / BMP\_HEIGHT][20];

            buff[y][x][2] = B[y\*height / BMP\_HEIGHT][20];

        }

    }

    //painting

    //--------------1、translation------------------//

    //translation(1, 1);

    //--------------2、rotation---------------------//

    //double angle;

    //angle = 3.1415926/6;

    //rotation(angle);

    //--------------3、Scale---------------------//

    //double coeffx,coeffy;//缩放倍数，放大则大于1，缩小则小于1

    //coeffx = 0.5;

    //coeffy = 0.8;

    //scale(coeffx,coeffy);

    //--------------4、Shear---------------------//

    //double kx,ky;//x方向错切系数与y方向错切系数

    //kx = 0.2;

    //ky = 0.8;

    //shear(kx,ky);

    //--------------5、Mirror---------------------//

    //mirror(1,1);//第一个1表示水平对称，第二个1表示垂直对称

    bmp\_create("test.bmp", (unsigned char \*)buff, BMP\_WIDTH, BMP\_HEIGHT, BMP\_PERW);

    return 0;

}

#### 2、目标函数

//平移函数

//将画布n\*n等分，new\_x与new\_y对应新坐标

void translation(int new\_x, int new\_y);

//旋转函数

//angle代表顺时针旋转angle

//angle是弧度制

void rotation(double angle);

//缩放

//coeffx与coeffy对应缩放倍数

//大于1时表示放大，小于1时表示缩小

void scale(double coeffx, double coeffy);

//错切

void shear(double k1, double k2);

//对称

//flag1为1时表示x方向对称

//flag2为1时表示y方向对称

void mirror(int flag1, int flag2);

2.1 图像的平移（Translation）

void translation(int new\_x, int new\_y) {

    int i0, j0;

    for (i0 = 0; i0 < width; i0++)

    {

        for (j0 = 0; j0 <height; j0++)

        {

            buff[new\_x\*height + j0][new\_y\*width + i0][0] = R[j0][i0];

            buff[new\_x\*height + j0][new\_y\*width + i0][1] = G[j0][i0];

            buff[new\_x\*height + j0][new\_y\*width + i0][2] = B[j0][i0];

        }

}

2.2 图像的旋转（Rotation）

void rotation(double angle) {

    double x1, y1, cos\_a, sin\_a, d1, d2, d;

    int m, n, color;

    double new\_width, new\_height;

    new\_width = width \* cos(angle) + height \* sin(angle);

    new\_height = height \* cos(angle) + width \* sin(angle);

    cos\_a = cos(angle);

    sin\_a = sin(angle);

    for (int j = 0; j < new\_height; j++) {

        for (int i = 0; i < new\_width; i++) {

            x1 = cos\_a \* (j - new\_height / 2) - sin\_a \* (i - new\_width / 2) + width / 2;

            y1 = sin\_a \* (j - new\_height / 2) + cos\_a \* (i - new\_width / 2) + height / 2;

            n = floor(x1);

            m = floor(y1);

            if (m >= 0 && n >= 0 && n < width&&m < height) {

                for (color = 0; color < 3; color++) {

                    switch (color){

                    case 0:

                        d1 = R[m][n] \* (n + 1 - x1) + R[m][n + 1] \* (x1-n);

                        d2 = R[m + 1][n] \* (n + 1 - x1) + R[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m+1- y1) + d2 \* (y1 -m);

                        break;

                    case 1:

                        d1 = G[m][n] \* (n + 1 - x1) + G[m][n + 1] \* (x1 - n);

                        d2 = G[m + 1][n] \* (n + 1 - x1) + G[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    default:

                        d1 = B[m][n] \* (n + 1 - x1) + B[m][n + 1] \* (x1 - n);

                        d2 = B[m + 1][n] \* (n + 1 - x1) + B[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    }

                    buff[j][i][color] =d;

                }

            }

        }

    }

}

2.3 图像的放缩（Scale）

void scale(double coeffx,double coeffy) {

    double x1, y1, d1, d2, d;

    int m, n;

    double new\_height, new\_width;

    new\_height = height \* coeffy;

    new\_width = width \* coeffx;

    for (int j = 0; j < new\_height; j++) {

        for (int i = 0; i < new\_width; i++) {

            y1 = j / coeffy;

            x1 = i / coeffx;

            m = floor(y1);

            n = floor(x1);

            if (m >= 0 && n >= 0 && n < width&&m < height) {

                for (int color = 0; color < 3; color++) {

                    switch (color) {

                    case 0:

                        d1 = R[m][n] \* (n + 1 - x1) + R[m][n + 1] \* (x1 - n);

                        d2 = R[m + 1][n] \* (n + 1 - x1) + R[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    case 1:

                        d1 = G[m][n] \* (n + 1 - x1) + G[m][n + 1] \* (x1 - n);

                        d2 = G[m + 1][n] \* (n + 1 - x1) + G[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    default:

                        d1 = B[m][n] \* (n + 1 - x1) + B[m][n + 1] \* (x1 - n);

                        d2 = B[m + 1][n] \* (n + 1 - x1) + B[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    }

                    buff[j][i][color] = d;

                }

            }

        }

    }

}

2.4 图像的错切（Shear）

void shear(double k1, double k2) {

    double x1, y1, d1, d2, d;

    int m, n;

    double new\_height, new\_width;

    new\_height = height+k2\*width;

    new\_width = width+k1\*height;

    for (int j = 0; j < new\_height; j++) {

        for (int i = 0; i < new\_width; i++) {

            y1 = (k2\*i - j) / (k1\*k2 - 1);

            x1 = (k1\*j - i) / (k1\*k2 - 1);

            m = floor(y1);

            n = floor(x1);

            if (m >= 0 && n >= 0 && n < width&&m < height) {

                for (int color = 0; color < 3; color++) {

                    switch (color) {

                    case 0:

                        d1 = R[m][n] \* (n + 1 - x1) + R[m][n + 1] \* (x1 - n);

                        d2 = R[m + 1][n] \* (n + 1 - x1) + R[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    case 1:

                        d1 = G[m][n] \* (n + 1 - x1) + G[m][n + 1] \* (x1 - n);

                        d2 = G[m + 1][n] \* (n + 1 - x1) + G[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    default:

                        d1 = B[m][n] \* (n + 1 - x1) + B[m][n + 1] \* (x1 - n);

                        d2 = B[m + 1][n] \* (n + 1 - x1) + B[m + 1][n + 1] \* (x1 - n);

                        d = d1 \* (m + 1 - y1) + d2 \* (y1 - m);

                        break;

                    }

                    buff[j][i][color] = d;

                }

            }

        }

    }

}

2.5 图像的镜像变化（Mirror）

void mirror(int flag1, int flag2) {

    if (flag1 == 1&&flag2==0) {

        for (int j = 0; j < height; j++) {

            for (int i = width; i < 2\*width; i++) {

                buff[j][i][0] = R[j][2 \* width - 1 - i];

                buff[j][i][1] = G[j][2 \* width - 1 - i];

                buff[j][i][2] = B[j][2 \* width - 1 - i];

            }

        }

    }

    if (flag2 == 1&&flag1==0) {

        for (int j = height; j < 2 \* height; j++) {

            for (int i = 0; i < width; i++) {

                buff[j][i][0] = R[2 \* height - 1 - j][i];

                buff[j][i][1] = G[2 \* height - 1 - j][i];

                buff[j][i][2] = B[2 \* height - 1 - j][i];

            }

        }

    }

    if (flag1 == 1 && flag2 == 1) {

        for (int j = height; j < 2 \* height; j++) {

            for (int i = width; i < 2\*width; i++) {

              buff[j][i][0] = R[2 \* height - 1 - j][2 \* width - 1 - i];

              buff[j][i][1] = G[2 \* height - 1 - j][2 \* width - 1 - i];

              buff[j][i][2] = B[2 \* height - 1 - j][2 \* width - 1 - i];

            }

        }

    }

}

#### 3、结果

【原图】

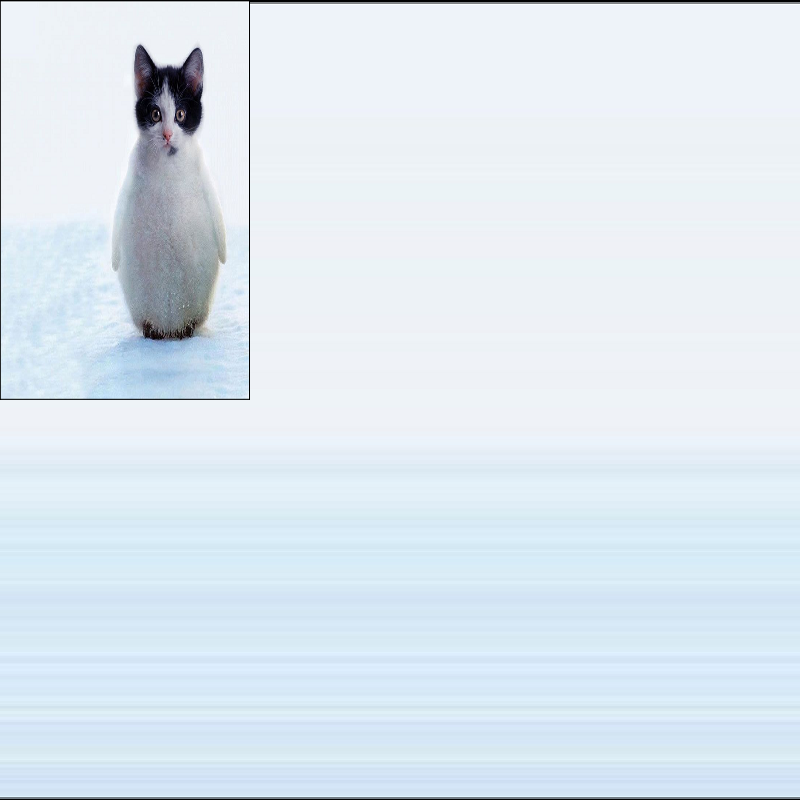


3.1图像的平移（Translation）【translation. bmp】

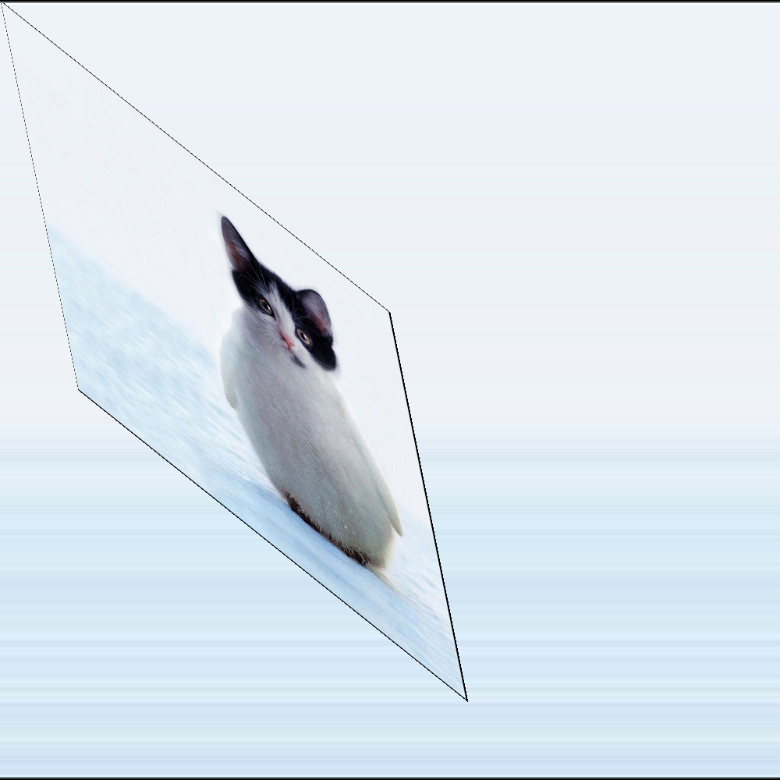
2、图像的旋转（Rotation）【rotation.bmp】



3、图像的放缩（Scale）【scale.bmp】



4、图像的错切（Shear）【shear.bmp】

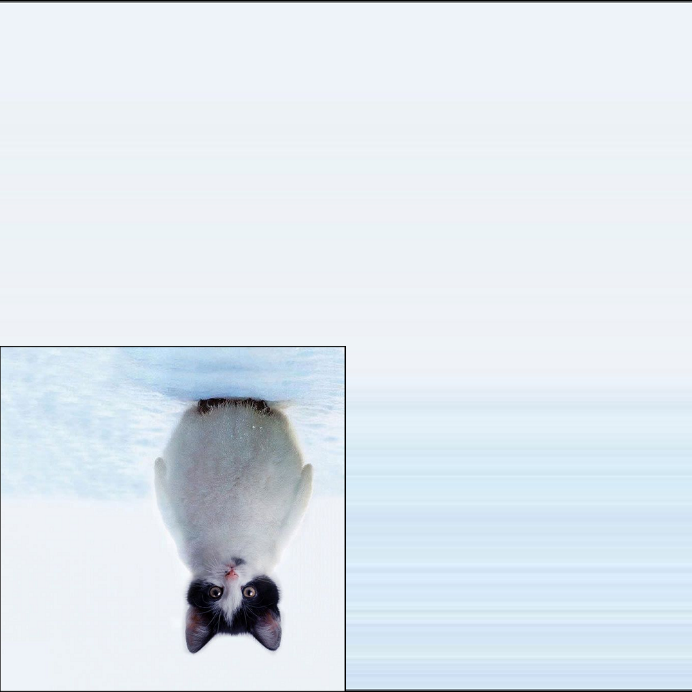


5、图像的镜像变化（Mirror）

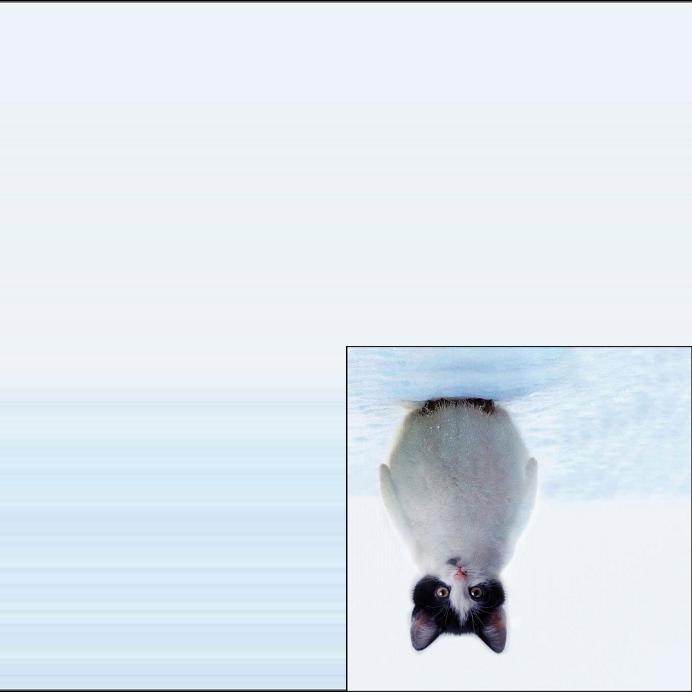
【mirror\_x.bmp】x轴方向对称



【mirror\_y.bmp】y轴方向对称



【mirror\_xy.bmp】中心对称



### 四、心得体悟

1、这次实验与前几次实验不同，画布尺寸与输入图间发生了变化，需要我们在创建图片时进行自定义，我在createBMP函数中被困扰了很久，多次输入乱序图片，存在问题是忽略了三维数组输出，把它直接视为二维数组会导致问题。

2、创建画布时要注意长宽像素值需要是4的倍数