

图像信息处理 - 实验4

黄昭阳 3130000696 - 12/11/15



接口解释

```
96 //Lab4 start
97 void fill(int row,int col);
98 void mapResultMatrixWithCanvasSize(float canvasScaleCoef);
99 void mirrorX();
100 void mirrorY();
101 void translate(int x,int y);
102 void scale(float x,float y);
103 void rotate(float degree);
104 void shear(float x,float y);
105 }
```

mirrorX(int x) —— 以 $X = x$ 为对称轴进行镜像映射

mirrorY(int y) —— 以 $Y = y$ 为对称轴进行镜像映射

translate(int x,int y) —— 图片移动的x、y个像素点

scale(float x,float y) —— 图片在x方向、y方向上的放缩系数

rotate(float degree) —— 图片以原点为中心逆时针旋转的角度

shear(float x,float y) —— 图片在x、y方向上的倾斜程度（即倾斜直线的系数）

mapResultMatrixWithCanvasSize(float canvasScaleCoef) —— 将原来的画布按照 canvasScaleCoef 进行放缩，并将原来的 bmp 数据按照之前做的变换映射到画布上，用来观察 translate 等的效果。

为了代码逻辑看起来正常点，我写了一个矩阵类，用来做矩阵乘法和生成这五种变换矩阵，代码如下，其中这五种转换矩阵均为标准的齐次坐标线性转换矩阵，这里不进行赘述，五种矩阵代码逻辑如下。

```
Matrix* Matrix::Translate(Matrix* m, float x, float y)
{
    if(m->Row !=3 )
    {
        #ifdef DEBUG
            cout << "Matrix::Translate" << endl;
            cout << " m.Row = " << m->Row << endl;
        #endif
        return NULL;
    }
    Matrix* translateMatrix = new Matrix(3,3);
    translateMatrix->data[0][0] = 1;
    translateMatrix->data[1][1] = 1;
    translateMatrix->data[2][2] = 1;    //对角线为1
    translateMatrix->data[0][2] = x;
    translateMatrix->data[1][2] = y;    //两个平移参数
    Matrix* resultMatrix = Matrix::Multiplication(translateMatrix,m);
    return resultMatrix;
}
```

```

Matrix* Matrix::MirrorX(Matrix* m, float x)
// 以X = x为轴进行对称变换的矩阵
{
    if(m->Row !=3 )
    {
        #ifdef DEBUG
            cout << "Matrix::MirrorX" << endl;
            cout << " m.Row = " << m->Row << endl;
        #endif
        return NULL;
    }
    Matrix* resultMatrix;
    Matrix* mirrorMatrix = new Matrix(3,3);
    mirrorMatrix->data[0][0] = 1;
    mirrorMatrix->data[1][1] = -1;
    mirrorMatrix->data[2][2] = 1;           //对应的参数

    mirrorMatrix = Matrix::Translate(mirrorMatrix,2*x,0);
    //mirrorMatrix是按照x=0为轴进行对称
    //变换的, 这里再进行一次平移来得到以X = x为轴的对称变换
    resultMatrix = Matrix::Multiplication(mirrorMatrix,m);
    return resultMatrix;
}

Matrix* Matrix::MirrorY(Matrix* m, float y)
// 以Y = y为轴进行对称变换的矩阵
{
    if(m->Row !=3 )
    {
        #ifdef DEBUG
            cout << "Matrix::MirrorY" << endl;
            cout << " m.Row = " << m->Row << endl;
        #endif
        return NULL;
    }
    Matrix* mirrorMatrix = new Matrix(3,3);
    Matrix* resultMatrix;

    mirrorMatrix->data[0][0] = -1;
    mirrorMatrix->data[1][1] = 1;
    mirrorMatrix->data[2][2] = 1;

    mirrorMatrix = Matrix::Translate(mirrorMatrix,0,2*y);
    resultMatrix = Matrix::Multiplication(mirrorMatrix,m);

    return resultMatrix;
}

```

```

Matrix* Matrix::Scale(Matrix* m, float xcoeff, float ycoeff)
{
    if(m->Row !=3 )
    {
        #ifdef DEBUG
            cout << "Matrix::Scale" << endl;
            cout << " m.Row = " << m->Row << endl;
        #endif
        return NULL;
    }
    Matrix* scaleMatrix = new Matrix(3,3);
    Matrix* resultMatrix;

    scaleMatrix->data[0][0] = xcoeff;
    scaleMatrix->data[1][1] = ycoeff;
    scaleMatrix->data[2][2] = 1;
    //放缩短矩阵的对应系数放置
    resultMatrix = Matrix::Multiplication(scaleMatrix,m);

    return resultMatrix;
}

Matrix* Matrix::Rotate(Matrix* m, float degree)
{
    if(m->Row !=3 )
    {
        #ifdef DEBUG
            cout << "Matrix::Rotate" << endl;
            cout << " m.Row = " << m->Row << endl;
        #endif
        return NULL;
    }
    Matrix* rotateMatrix = new Matrix(3,3);
    Matrix* resultMatrix;

    rotateMatrix->data[0][0] = rotateMatrix->data[1][1] = cos(degree);
    rotateMatrix->data[0][1] = -sin(degree);
    rotateMatrix->data[1][0] = sin(degree);
    rotateMatrix->data[2][2] = 1;
    //旋转矩阵的对应系数放置

    resultMatrix = Matrix::Multiplication(rotateMatrix,m);

    return resultMatrix;
}

```

```

Matrix* Matrix::Shear(Matrix* m, float x, float y)
{
    if(m->Row !=3 )
    {
        #ifdef DEBUG
            cout << "Matrix::Shear" << endl;
            cout << " m.Row = " << m->Row << endl;
        #endif
        return NULL;
    }
    Matrix* rotateMatrix = new Matrix(3,3);
    Matrix* resultMatrix;

    rotateMatrix->data[0][0] = rotateMatrix->data[1][1] = 1;
    rotateMatrix->data[0][1] = x;
    rotateMatrix->data[1][0] = y;
    rotateMatrix->data[2][2] = 1;

    resultMatrix = Matrix::Multiplication(rotateMatrix,m);

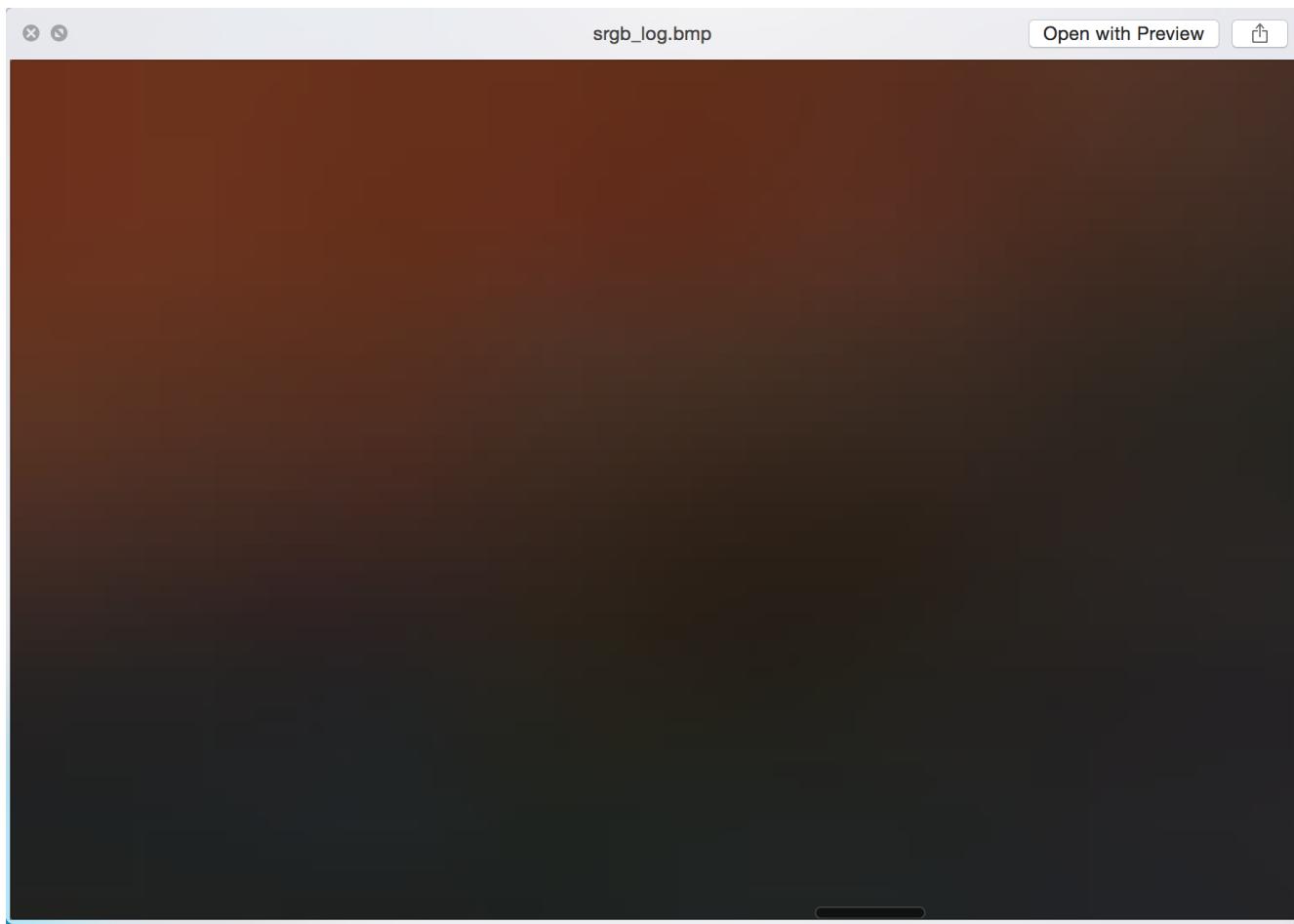
    return resultMatrix;
}

```

上面的系数设置是按照老师的PPT上来的，但是PPT上的mirror矩阵存在明显问题，第二行第三列的系数应当是0而不是1，这一点一开始没注意导致我求逆矩阵的时候出现错误，找了很久。

代码的使用是：

1. 可以不断使用以上五个操作对图片进行线性变换，这个时候我只是维护一个3*3的变换矩阵，而不是真将整张图片全部转换，因此消耗时间很少。
2. 使用mapResultMatrixWithCanvasSize(float canvasScaleCoef)函数将图片按照要求画到画布上，这个时候将画布的所有坐标放到一个3*n的矩阵中，利用逆矩阵算出每个像素点在原来bmp图片上的坐标，并且利用双线性插值来计算这个点的像素值，一开始写了一个梯度法解线性方程组来解，后来发现不对，因为我x,y使用的坐标直接是像素坐标，这样图片位置不同，方程的系数也会不一样，这样明显是不正常的，后来发现其实x,y就是包围被计算素点周围的四个像素点的相对坐标值，即(0,0),(0,1),(1,0),(1,1)，因此方程产生退化，可以直接解出来带进去就可以了，下面是双线性插值的一张显示图。



这是封面图放大五倍，并且用图片显示器再不断拉大后的显示结果，效果明显，边缘被融合。

以下是效果展示图。

一、展示平移效果：

将原图平移宽度和高度长，然后画布放大三倍，这样图片应当在画布正中央。

```
    bmpFile* bf = new bmpFile(fileName);
// bf->rotate(30.0/180.0*PI);
// bf->mirrorX();
//bf->shear(1,0);
bf->translate(bf->bmpInfoHeader.biWidth, bf->bmpInfoHeader.biHeight);
bf->mapResultMatrixWithCanvasSize(3);
bf->exportToFile(fileout);
```



二、展示旋转效果:

将图片旋转30°后放到三倍放大画布的正中央。

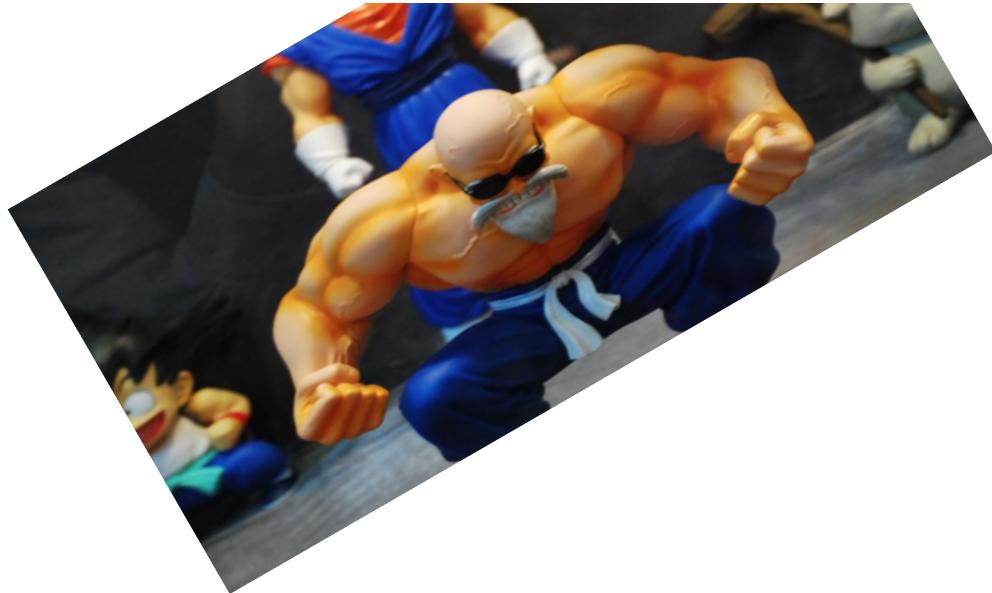
```
54
35     bmpFile* bf = new bmpFile(fileName);
36     bf->rotate(30.0/180.0*PI);
37 //   bf->mirrorX();
38 //   bf->shear(1,0);
39     bf->translate(bf->bmpInfoHeader.biWidth,bf->bmpInfoHeader.biHeight);
40     bf->mapResultMatrixWithCanvasSize(3);
41     bf->exportToFile(fileout);
42 }
```



三、拉伸效果展示：

先进行长宽比不一样的拉伸，然后旋转，再平移到中间，由于先旋转后放缩会产生图片的形变，这里先使用放缩以便观察放缩效果，其他的组合方式在后面放出。

```
35     bmpFile* bf = new bmpFile(fileName);
36     bf->scale(2,1.5);
37     bf->rotate(30.0/180.0*PI);
38 //   bf->mirrorX();
39 //   bf->shear(1,0);
40     bf->translate(bf->bmpInfoHeader.biWidth,bf->bmpInfoHeader.biHeight);
41     bf->mapResultMatrixWithCanvasSize(3);
42     bf->exportToFile(fileout);
```



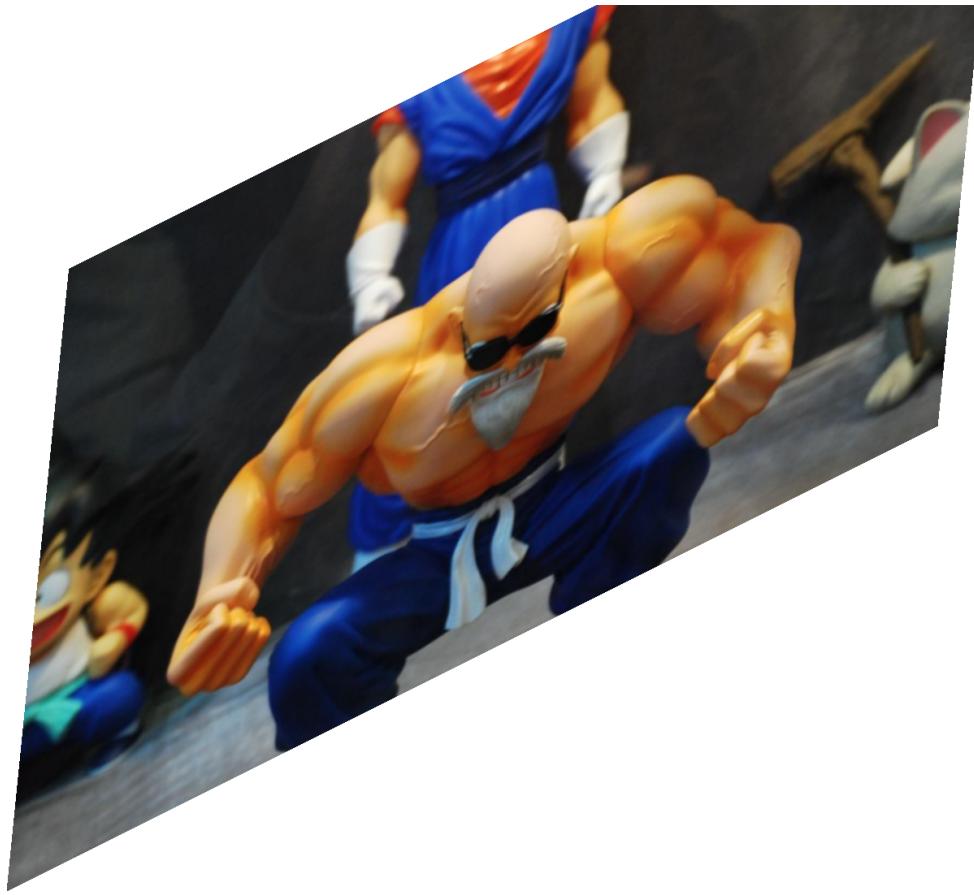
四、镜像效果，分为x和y两个方向的镜像：按照宽度轴镜像一次，然后画布扩展三倍

```
35  bmpFile* bf = new bmpFile(fileName);
36 // bf->scale(2,1.5);
37 // bf->rotate(30.0/180.0*PI);
38  bf->mirrorX(bf->bmpInfoHeader.biWidth);
39 // bf->mirrorX();
40 //bf->shear(1,0);
41 // bf->translate(bf->bmpInfoHeader.biWidth,bf->bmpInfoHeader.biHeight);
42  bf->mapResultMatrixWithCanvasSize(3);
43  bf->exportToFile(fileout);
```



五、shear效果展示图：两条倾斜直线系数分别为0.1和0.5。

```
42     bf->shear(0.1,0.5);  
43     bf->mapResultMatrixWithCanvasSize(1.5);
```



六、 mapResultMatrixWithCanvasSize(float canvasScaleCoef)的实现

```
newWidth = bmpInfoHeader.biWidth*canvasScaleCoef;
newHeight = bmpInfoHeader.biHeight*canvasScaleCoef; //得到新的画布数据

Matrix* resultPos = new Matrix(3,newHeight*newWidth);
Matrix* originalPos;
int row,col;
for(int i=0;i<newHeight;i++)
    for(int j=0;j<newWidth;j++)
    {
        resultPos->data[0][i*newWidth+j] = j;
        resultPos->data[1][i*newWidth+j] = i;
        resultPos->data[2][i*newWidth+j] = 1;
    }
//将所有的现在画布中坐标值都放入一个3*N的矩阵中，用逆矩阵求其原来的坐标
```

```

originalPos =
Matrix::Multiplication(reversTransformMatrix, resultPos);
//reversTransformMatrix为线性转换矩阵的逆矩阵，用矩阵乘法求原画布坐标

for(int i=0;i<newHeight;i++)
    for(int j=0;j<newWidth;j++)
    {
        row = int(originalPos->data[1][i*newWidth+j]);
        col = int(originalPos->data[0][i*newWidth+j]);
        x = originalPos->data[0][i*newWidth+j]-col;
        y = originalPos->data[1][i*newWidth+j]-row;
        rown = row+1; coln = col+1;
        if(rown>=oriHeight) rown = row;
        if(coln>=oriWidth) coln = col;
        for(int k=0;k<3;k++)
            if(row<0||col<0||row>=oriHeight||col>=oriWidth)
data[i][j*3+k] = 255; //不在画布中的坐标染成白色
        else
        {
            d = bmpFileData[row][col*3+k];
            a = bmpFileData[row][coln*3+k]-d;
            b = bmpFileData[rown][col*3+k]-d;
            c = bmpFileData[rown][coln*3+k]-a-b-d;
            d = a*x+b*y+c*x*y+d;
        //画布内的坐标用双线性插值求出
            if(d>255)
                data[i][j*3+k] = 255;
            else if(d<0)
                data[i][j*3+k] = 0;
            else data[i][j*3+k] = int(d);
        //将超出值域的值clip(因为原来几个系数之和并不为1)
        }
    }
}

//声明，源代码中有大段注释，是因为原本做的是画布会随着图像的移动而移动，而这样就看不出
translate效果了，这里改用扩展画布尺寸来展示translate效果

```

七、其他的一下奇奇怪怪的效果图

