浙江大学实验报告

专业：计算机科学与技术
姓名：NoughtQ
学号：1145141919810
日期：2024 年 11 月 20 日

课程名称：图像信息处理　　指导老师：宋明黎　　成绩：_____

实验名称：图像的基本几何变换

一、 实验目的和要求

实现一些简单的图像几何变换，包括以下操作：

- 平移操作
- 旋转操作
- 缩放操作
- 错切操作
- 镜像操作

二、 实验内容和原理

2.1 几何变换

几何变换(geometric transform)是指利用变换函数改变图像中像素的位置，从而产生新图像的过程。几何变换不改变像素值，而是改变像素所在的位置。

它的一般方程为：

$$g(x,y)=f(x',y')=f[a(x,y),b(x,y)]$$

其中$f(x,y)$表示输入图像，$g(x,y)$表示输出图像，函数$a(x,y),b(x,y)$唯一地描述了空间变换。

几何变换根据难易程度通常可以归结为两类：

- **简单变换**(simple transformation)：变换过程（各个像素变换前后的位置）以及变换参数可知时的变换，如图像的平移、镜像、转置、旋转、缩放、错切变换等。
- **一般变换**(general transformation)：变换过程不是一目了然，变换参数难以测量时的变换。通常情况下，对图像畸变进行校正时，需要用到较为复杂的变换公式。

下面介绍的变换均为简单变换。

2.1.1 平移

平移(translation)：将图像沿水平和竖直方向移动，从而产生新图像的过程。它的方程为：

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

可以看出，该方程使得原图像中的每个像素沿x方向平移x_0，沿y方向平移y_0。

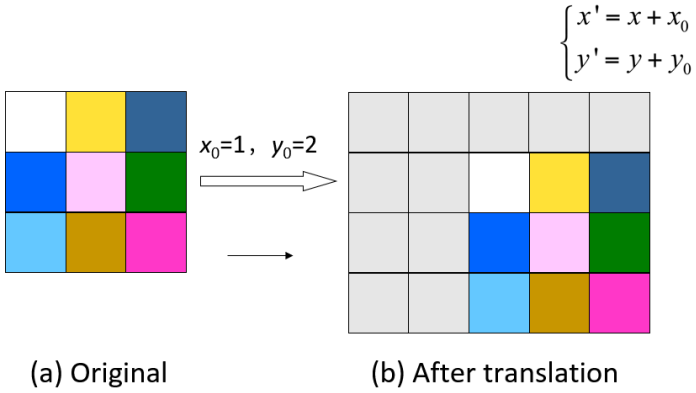


Figure 1: 平移变换

注意：虽然平移后的景物与原图像相同，但“画布(canvas)”的面积可能需要扩大，否则就会丢失信息。

2.1.2 旋转

旋转(rotation)：绕原点旋转θ角，得到新图像的过程。它的方程为：

$$x' = x\cos\theta - y\sin\theta, y' = x\sin\theta + y\cos\theta$$

空洞问题

图像经过旋转变换以后，新图像中会出现许多空洞，如下图所示：

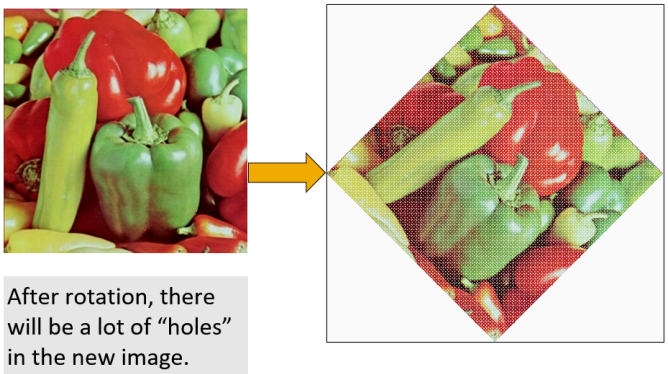


Figure 2: 旋转变换

解决方法

用**插值** (interpolation)方法填补：按顺序寻找每一行中的空洞像素，设置其像素值与同一行中前一个像素的像素值相同（行插值）。

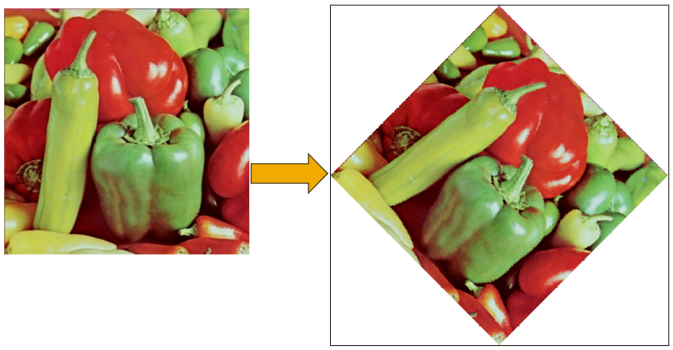


Figure 3: 用了插值法后的旋转变换

现在旋转后的图像如下所示，可以看到空洞已经被填上了。

2.1.3 缩放

缩放(scale)：将图像乘以一定系数，从而产生新图像的过程。它的方程为：

$$\begin{cases} x' = cx \\ y' = cy \end{cases}$$

- 沿 x 轴方向缩放c倍（$c > 1$时为放大，$0 < c < 1$时为缩小）；沿 y 轴方向缩放d倍（$d > 1$时为放大，$0 < d < 1$时为缩小）
- 当$c = d$时，图像等比缩放；否则为非等比缩放，导致图像变形

缩放的两种类型：

- 缩小(shrink)：按一定间隔选取某些行和列的像素构成缩小后的新图像
- 放大(enlarge)：新图像出现空行和空列，可采用插值的方法加以填补，但存在“马赛克”现象

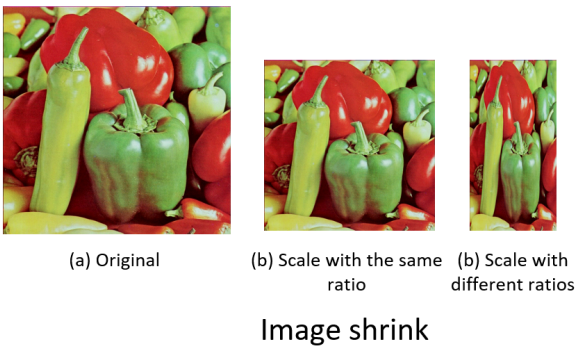


Figure 4: 缩放变换（缩小）

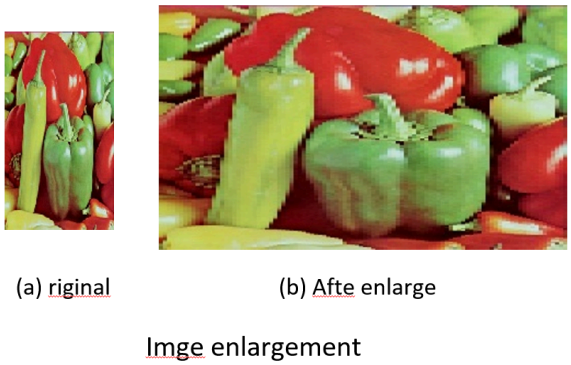


Figure 5: 缩放变换（放大）

2.1.4 错切

错切(shear)：景物在平面上的非垂直投影效果。它的方程为：

$$\text{Shear on x axis: } \begin{cases} a(x,y) = x + d_x y \\ b(x,y) = y \end{cases}$$

$$\text{Shear on y axis: } \begin{cases} a(x,y) = x \\ b(x,y) = y + d_y x \end{cases}$$

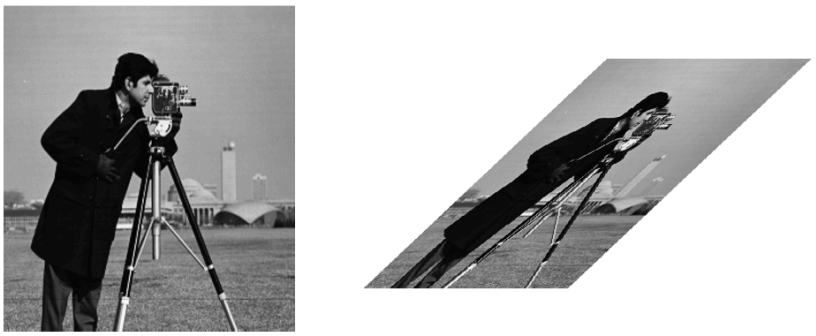


Figure 6: 错切变换

2.1.5 镜像

镜像(mirror)：将图像绕x轴或y轴翻转，从而产生与原图像对称的新图像的过程。它的方程为：

$$\text{Flip around x axis: } \begin{cases} x' = x \\ y' = -y \end{cases}$$

$$\text{Flip around y axis: } \begin{cases} x' = -x \\ y' = y \end{cases}$$

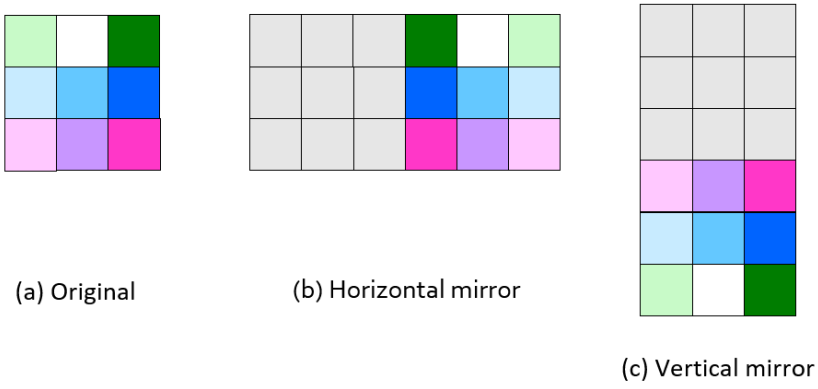


Figure 7: 镜像变换

2.2 插值

插值(interpolation)：几何变换最常用的工具，利用已知像素值，采用不同的插值方法，可以模拟出未知像素的像素值。有以下几种插值方法：

- **最近邻插值**(nearest neighbor interpolation)
- **线性插值**(linear interpolation)
- **径向基函数插值**(RBF interpolation)

2.2.1 最近邻插值

最近邻插值：输出像素的灰度值等于离它所映射到的位置最近的输入像素的灰度值。

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

用数学语言表示上述过程：

$$(x', y') \xrightarrow{\text{inverse transformation}} (x, y) \xrightarrow{\text{rounding operation}} (x_{\text{int}}, y_{\text{int}})$$
$$\xrightarrow{\text{assign value}} I_{\text{new}}(x', y') = I_{\text{old}}(x_{\text{int}}, y_{\text{int}})$$

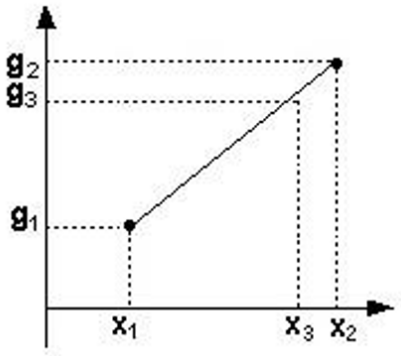
> **局限性**
>
> 当图像中包含明显的几何结构时，结果将不太光滑连续，从而在图像中产生人为的痕迹。

2.2.2 线性插值

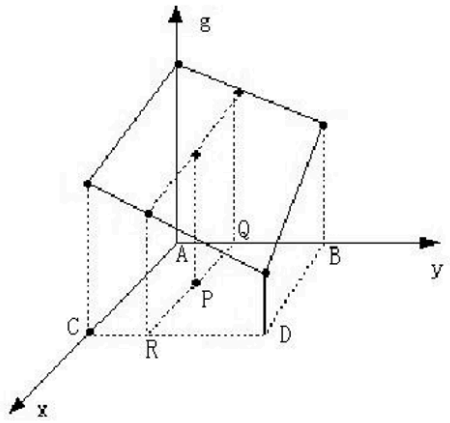
线性插值的几种情况：

- 一维：已知x_1和x_2处的灰度值分别为g_1和g_2，则x_3处的灰度值g_3为：

$$g_3 = \frac{g_2 - g_1}{x_2 - x_1}(x_3 - x_1) + g_1$$



- 二维（**双线性插值**(bilinear interpolation)）：已知图像的正方形网格上四个点A、B、C、D的灰度，求P点的灰度。



步骤如下：

- 定义双线性方程$g(x,y)=ax+by+cxy+d$
- 分别将A、B、C、D四点的位置和灰度代入方程，得到方程组
- 解方程组，解出a、b、c、d四个系数
- 将P点的位置代入方程，得到P点的灰度

2.2.3 径向基函数插值

径向基函数插值(radial basis function based interpolation)，它的公式为：

$$G(x)=\sum_{i=1}^{n}w_iG(c_i)\quad \text{where } w_i=\frac{\varphi(|x-c_i|)}{\sum_{\{i=1\}}^{n}\varphi(|x-c_i|)}$$

其中x可以是一个标量，也可以是一个向量。因此此类插值法既可以作为一维插值，也可以作为二维、多维插值，这取决于x的维度。

RBF 插值法的核心是关于半径r的核函数$\varphi(r)$，常用的核函数有（前两种函数都有一个调节参数σ）：

- 高斯(Gaussian)：$\varphi(r)=\exp\left(-\frac{r^2}{2\sigma^2}\right)$
- 多元二次(multiquadrics)：$\varphi(r)=\sqrt{1+\frac{r^2}{\sigma^2}}$
- 线性：$\varphi(r)=r$
- 三次：$\varphi(r)=r^3$
- 薄板(thinplate)：$\varphi(r)=r^2ln(r+1)$

三、 实验步骤与分析

3.1 平移

与平移操作相关的代码如下所示:

```
int TransX, TransY;                    // 实际平移量

// 获取平移量
void getTransData() {
    std::string s;

    printf("Do you want to customize the translation data?\n");
    printf("Please input yes or no: \n");
    std::cin >> s;
    if (s[0] == 'y' || s[0] == 'Y') {
       printf("Please input your expected translation data(integers):
\n");
         printf("Note that if you give a invalid input, the program
will terminate!\n");
        printf("X: ");
        scanf("%d", &TransX);
        printf("Y: ");
        scanf("%d", &TransY);
    } else if (s[0] == 'n' || s[0] == 'N') {
        TransX = TRANSLATIONX;
        TransY = TRANSLATIONY;
    } else {
        printf("Invalid choice, try again!\n");
        exit(1);
    }
}

// 平移操作
BMPFILE Translation(BMPFILE bf) {
    BMPFILE newImg;
    int i, x, y;
    int deltaX, deltaY;     // 计算用的平移量
    LONG width, height;
    int newColorBitWidth;
    int oldPos, newPos;

    // 分配空间
```

```
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(newImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(newImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

    // 获取平移量
    getTransData();

    // 根据平移量修改图像参数，对“画布”的大小做适当的调整
    newImg->bmih.biWidth += abs(TransX);
    newImg->bmih.biHeight += abs(TransY);
    width = newImg->bmih.biWidth;
    height = newImg->bmih.biHeight;
    newColorBitWidth = (width * RGBNUM + RGBNUM) / 4 * 4;
    newImg->bmih.biSizeImage = newColorBitWidth * height;
    newImg->bmfh.bfSize =
                  newImg->bmih.biSizeImage + newImg->bmfh.bfOffBits;
    newImg->aBitmapBits =
            (BYTE *)malloc(sizeof(BYTE) * newImg->bmih.biSizeImage);

    // 先让图像全白
    for (i = 0; i < newImg->bmfh.bfSize; i++)
        newImg->aBitmapBits[i] = 255;

    // 平移图像
    deltaX = max(0, TransX);
    deltaY = max(0, TransY);
    for (y = 0; y < bf->bmih.biHeight; y++)
        for (x = 0; x < bf->bmih.biWidth; x++) {
            oldPos = y * ColorBitWidth + x * 3;
            newPos = (y + deltaY) * newColorBitWidth
                        + (x + deltaX) * 3;
            newImg->aBitmapBits[newPos] =
                                        bf->aBitmapBits[oldPos];
            newImg->aBitmapBits[newPos + 1] =
                                        bf->aBitmapBits[oldPos + 1];
            newImg->aBitmapBits[newPos + 2] =
                                        bf->aBitmapBits[oldPos + 2];
        }
```

```
    return newImg;
}
```

平移的具体实现过程为：

1. 拷贝旧图的文件头部分到新图
2. 获取平移量（用户可以自行设置，也可以直接使用默认值）
3. 根据平移量调整新图的画布大小，以便容纳平移后的图像（需要调整相关文件头字段值）
4. 令图像所有像素为白色，然后通过平移所有的像素点来平移整个图像

3.2 旋转

与旋转操作相关的代码如下所示：

```
int Degree;                                    // 旋转角度（角度制）

// 获取旋转量
void getRotData(double * sinR, double * cosR) {
    std::string s;
    double radian;

    printf("Do you want to set the rotation degree value by yourself?
\n");
    printf("Please input yes or no: \n");
    std::cin >> s;
    if (s[0] == 'y' || s[0] == 'Y') {
        printf("Please input your expected rotation degree(integers,
degree notation):\n");
         printf("Note that if you give a invalid input, the program
will terminate!\n");
        printf("Degree: ");
        scanf("%d", &Degree);
    } else if (s[0] == 'n' || s[0] == 'N') {
        Degree = DEFAULTDEGREE;
    } else {
        printf("Invalid choice, try again!\n");
        exit(1);
    }

    radian = Degree * M_PI / 180;
    *sinR = sin(radian);
    *cosR = cos(radian);
}
```

```
// 旋转像素的 x 值
int rotationX(int x, int y, double sinR, double cosR) {
    return round(x * cosR - y * sinR);
}

// 旋转像素的 y 值
int rotationY(int x, int y, double sinR, double cosR) {
    return round(x * sinR + y * cosR);
}

// 旋转操作
BMPFILE Rotation(BMPFILE bf) {
    BMPFILE newImg;
    int i, x, y;
    int deltaX, deltaY;         // 中心点的平移量
    int tmpX, tmpY;
    int oldX, oldY;             // 最近邻插值计算时得到的理论上原图像素位置
    LONG oldWidth, oldHeight;
    LONG width, height;
    int newColorBitWidth;
    int newPos, oldPos;
    double sinR, cosR;          // 计算旋转所需的三角函数

    // 分配空间
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(newImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(newImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

    // 获取旋转参数
    getRotData(&sinR, &cosR);

    // 根据旋转角度修改图像参数，对“画布”的大小做适当的调整
    oldWidth = bf->bmih.biWidth;
    oldHeight = bf->bmih.biHeight;
    // 这里根据几何关系计算画布的大小
    newImg->bmih.biWidth =
              round(oldWidth * fabs(cosR) + oldHeight * fabs(sinR));
    newImg->bmih.biHeight =
              round(oldWidth * fabs(sinR) + oldHeight * fabs(cosR));
```

```
    width = newImg->bmih.biWidth;
    height = newImg->bmih.biHeight;
    newColorBitWidth = (width * RGBNUM + RGBNUM) / 4 * 4;
    newImg->bmih.biSizeImage = newColorBitWidth * height;
    newImg->bmfh.bfSize =
                  newImg->bmih.biSizeImage + newImg->bmfh.bfOffBits;
    newImg->aBitmapBits =
            (BYTE *)malloc(sizeof(BYTE) * newImg->bmih.biSizeImage);

    // 计算中心点偏移量，相当于将旋转转化成平移操作
    // 便于后面用最近邻插值法补上空洞
    deltaX = width / 2
              - rotationX(oldWidth / 2, oldHeight / 2, sinR, cosR);
    deltaY = height / 2
              - rotationY(oldWidth / 2, oldHeight / 2, sinR, cosR);

    // 旋转图像
    for (y = 0; y < height; y++)
        for (x = 0; x < width; x++) {
            newPos = y * newColorBitWidth + x * 3;

            // 最近邻插值（做逆变换，先平移再旋转）
            tmpX = x - deltaX;
            tmpY = y - deltaY;
            // 原来旋转 degree 度，逆变换旋转 -degree 度
            // 此时 sinR 取反，cosR 不变
            oldX = round(tmpX * cosR + tmpY * sinR);
            oldY = round(-tmpX * sinR + tmpY * cosR);

            // 如果计算出来的点不在原图范围内，则用白色像素替代
            if (oldX < 0 || oldX >= oldWidth
                  || oldY < 0 || oldY >= oldHeight) {
                newImg->aBitmapBits[newPos] = 255;
                newImg->aBitmapBits[newPos + 1] = 255;
                newImg->aBitmapBits[newPos + 2] = 255;
                continue;
            }
            // 完成该像素点的旋转计算
            oldPos = oldY * ColorBitWidth + oldX * 3;
            newImg->aBitmapBits[newPos] =
                                              bf->aBitmapBits[oldPos];
            newImg->aBitmapBits[newPos + 1] =
```

```
                                            bf->aBitmapBits[oldPos + 1];
            newImg->aBitmapBits[newPos + 2] =
                                            bf->aBitmapBits[oldPos + 2];
        }

    return newImg;
}
```

旋转的具体实现过程为：

1. 拷贝旧图的文件头部分到新图
2. 获取旋转角度（用户可以自行设置，也可以直接使用默认值），并计算正弦值和余弦值
3. 根据旋转角度调整新图的画布大小（根据几何关系计算），以便容纳旋转后的图像（需要调整相关文件头字段值）
4. 由于旋转是以图像中心点为转轴的，且画布扩展的过程时中心点的位置会发生变化，所以需要计算中心点的偏移量（因此实际上旋转操作还包括平移操作）
5. 通过**最近邻插值**来实现具体的旋转操作，以填补旋转带来的空洞，具体过程为：
   - 遍历旋转后图像的每个像素点，对于每个像素点，通过逆变换（在旋转操作中，需要先平移后旋转）得到它在原图上的大致位置
   - 如果这个计算结果超出原图的范围，则用白色像素表示该像素点，否则的话就用原图上的对应像素（刚刚算出来的像素）值作为该像素点的值

> **注意**
>
> 之后的几个操作都会用到**最近邻插值**，具体实现过程与上述类似，故后面不会详细展开叙述。

3.3 缩放

与缩放操作相关的代码如下所示：

```
double ScaleX, ScaleY;                  // 缩放倍率

// 获取缩放倍率
void getScaleData() {
    std::string s;

    printf("Do you want to customize the scale data?\n");
    printf("Please input yes or no: \n");
    std::cin >> s;
    if (s[0] == 'y' || s[0] == 'Y') {
        printf("Please input your expected scale data(floating-point
```

```
number):\n");
         printf("Note that if you give a invalid input, the program
will terminate!\n");
        printf("X: ");
        scanf("%lf", &ScaleX);
        printf("Y: ");
        scanf("%lf", &ScaleY);
    } else if (s[0] == 'n' || s[0] == 'N') {
        ScaleX = DEFAULTSCALEX;
        ScaleY = DEFAULTSCALEY;
    } else {
        printf("Invalid choice, try again!\n");
        exit(1);
    }
}

// 缩放操作
BMPFILE Scale(BMPFILE bf) {
    BMPFILE newImg;
    int i, x, y;
    int oldX, oldY;             // 最近邻插值计算时得到的理论上原图像素位置
    LONG width, height;
    LONG oldWidth, oldHeight;
    int newColorBitWidth;
    int newPos, oldPos;

    // 分配空间
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(newImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(newImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

    // 获取缩放参数
    getScaleData();

    // 根据缩放倍率修改图像参数，对“画布”的大小做适当的调整
    oldWidth = bf->bmih.biWidth;
    oldHeight = bf->bmih.biHeight;
    newImg->bmih.biWidth = round(newImg->bmih.biWidth * ScaleX);
    newImg->bmih.biHeight = round(newImg->bmih.biHeight * ScaleY);
    width = newImg->bmih.biWidth;
```

```
    height = newImg->bmih.biHeight;
    newColorBitWidth = (width * RGBNUM + RGBNUM) / 4 * 4;
    newImg->bmih.biSizeImage = newColorBitWidth * height;
    newImg->bmfh.bfSize =
                newImg->bmih.biSizeImage + newImg->bmfh.bfOffBits;
    newImg->aBitmapBits =
            (BYTE *)malloc(sizeof(BYTE) * newImg->bmih.biSizeImage);

    // 缩放图像（还是使用最近邻插值处理空洞）
    for (y = 0; y < height; y++)
        for (x = 0; x < width; x++) {
            newPos = y * newColorBitWidth + x * 3;
            oldX = round(x / ScaleX);
            oldY = round(y / ScaleY);

            // 如果计算出来的点不在原图范围内，则用白色像素替代
            if (oldX < 0 || oldX >= oldWidth
              || oldY < 0 || oldY >= oldHeight) {
                newImg->aBitmapBits[newPos] = 255;
                newImg->aBitmapBits[newPos + 1] = 255;
                newImg->aBitmapBits[newPos + 2] = 255;
                continue;
            }

            oldPos = oldY * ColorBitWidth + oldX * 3;
            newImg->aBitmapBits[newPos] = bf->aBitmapBits[oldPos];
            newImg->aBitmapBits[newPos + 1]
                                    = bf->aBitmapBits[oldPos + 1];
            newImg->aBitmapBits[newPos + 2]
                                    = bf->aBitmapBits[oldPos + 2];
        }

    return newImg;
}
```

缩放的具体实现过程为：

1. 拷贝旧图的文件头部分到新图
2. 获取缩放倍率（同时设置 x 方向和 y 方向的缩放倍率；用户可以自行设置，也可以直接使用默认值）
3. 根据缩放倍率调整新图的画布大小，以便适应旋转后的图像（需要调整相关文件头字段值）
4. 通过最近邻插值来实现具体的缩放操作，以填补缩放过程带来的空洞

3.4 错切

与错切操作相关的代码如下所示:

```
// 错切的选择：0 表示沿 x 方向错切，1 表示沿 y 方向错切
int SheerChoice;
double SheerX, SheerY;                    // 错切倍率

// 获取错切数据
void getSheerData() {
    char c;
    std::string s;

    printf("Which sheer axis do you want to choose?\n");
   printf("Please input x or y, otherwise the program will terminate!
\n");
    printf("Your choice: ");
    getchar();
    c = getchar();
    if (c == 'x' || c == 'X') {
        SheerChoice = 0;
    } else if (c == 'y' || c == 'Y') {
        SheerChoice = 1;
    } else {
        printf("Invalid choice, try again!\n");
        exit(1);
    }

    printf("Do you want to customize the sheer data?\n");
    printf("Please input yes or no: \n");
    std::cin >> s;
    if (s[0] == 'y' || s[0] == 'Y') {
       printf("Please input your expected sheer data(floating-point
number):\n");
        printf("Note that if you give a invalid input, the program
will terminate!\n");
        if (c == 'x' || c == 'X') {
            printf("X: ");
            scanf("%lf", &SheerX);
        } else {
            printf("Y: ");
            scanf("%lf", &SheerY);
        }
```

```
    } else if (s[0] == 'n' || s[0] == 'N') {
        if (c == 'x' || c == 'X') {
            SheerX = DEFAULTSHEERX;
        } else {
            SheerY = DEFAULTSHEERY;
        }
    } else {
        printf("Invalid choice, try again!\n");
        exit(1);
    }
}

// 错切操作
BMPFILE Sheer(BMPFILE bf) {
    BMPFILE newImg;
    int i, x, y;
    int oldX, oldY;             // 最近邻插值计算时得到的理论上原图像素位置
    LONG width, height;
    LONG oldWidth, oldHeight;
    int newColorBitWidth;
    int newPos, oldPos;

    // 分配空间
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(newImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(newImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

    // 获取错切参数
    getSheerData();

    // 根据错切方向的选择和倍率修改图像参数，对“画布”的大小做适当的调整
    oldWidth = bf->bmih.biWidth;
    oldHeight = bf->bmih.biHeight;
    newImg->bmih.biWidth = !SheerChoice ?
                   oldWidth + round(SheerX * oldHeight) : oldWidth;
    newImg->bmih.biHeight = SheerChoice ?
                   oldHeight + round(SheerY * oldWidth) : oldHeight;
    width = newImg->bmih.biWidth;
    height = newImg->bmih.biHeight;
```

```
    newColorBitWidth = (width * RGBNUM + RGBNUM) / 4 * 4;
    newImg->bmih.biSizeImage = newColorBitWidth * height;
    newImg->bmfh.bfSize
              = newImg->bmih.biSizeImage + newImg->bmfh.bfOffBits;
    newImg->aBitmapBits
        = (BYTE *)malloc(sizeof(BYTE) * newImg->bmih.biSizeImage);

    // 错切图像（还是使用最近邻插值处理空洞）
    for (y = 0; y < height; y++)
        for (x = 0; x < width; x++) {
            newPos = y * newColorBitWidth + x * 3;

            oldX = !SheerChoice ? x - round(SheerX * y) : x;
            oldY = SheerChoice ? y - round(SheerY * x) : y;

            // 如果计算出来的点不在原图范围内，则用白色像素替代
            if (oldX < 0 || oldX >= oldWidth
              || oldY < 0 || oldY >= oldHeight) {
                newImg->aBitmapBits[newPos] = 255;
                newImg->aBitmapBits[newPos + 1] = 255;
                newImg->aBitmapBits[newPos + 2] = 255;
                continue;
            }

            oldPos = oldY * ColorBitWidth + oldX * 3;
            newImg->aBitmapBits[newPos] = bf->aBitmapBits[oldPos];
            newImg->aBitmapBits[newPos + 1]
                                    = bf->aBitmapBits[oldPos + 1];
            newImg->aBitmapBits[newPos + 2]
                                    = bf->aBitmapBits[oldPos + 2];
        }

    return newImg;
}
```

错切的具体实现过程为：

1. 拷贝旧图的文件头部分到新图
2. 获取错切方向（x 方向/y 方向）和倍率（用户可以自行设置，也可以直接使用默认值）
3. 根据错切方向和倍率调整新图的画布大小，以便容纳错切后的图像（需要调整相关文件头字段值）
4. 通过最近邻插值来实现具体的错切操作，以填补错切带来的空洞

3.5 镜像

与镜像操作相关的代码如下所示:

```
int MirrorChoice;                                    // 镜像的选择：0 表示沿 x 轴镜像，1
表示沿 y 轴镜像

// 获取镜像选择
void getMirrorData() {
    char c;

    printf("Which mirror axis do you want to choose?\n");
   printf("Please input x or y, otherwise the program will terminate!
\n");
    printf("Your choice: ");
    getchar();
    c = getchar();
    if (c == 'x' || c == 'X') {
        MirrorChoice = 0;
    } else if (c == 'y' || c == 'Y') {
        MirrorChoice = 1;
    } else {
        printf("Invalid choice, try again!\n");
        exit(1);
    }
}

// 镜像操作
BMPFILE Mirror(BMPFILE bf) {
    BMPFILE newImg;
    int i, x, y;
    int oldX, oldY;                 // 最近邻插值计算时得到的理论上原图像素位置
    LONG width, height;
    int newPos, oldPos;

    // 分配空间
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 将旧图的数据拷贝到新图上
    memcpy(&(newImg->bmfh), &(bf->bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(newImg->bmih), &(bf->bmih), sizeof(BITMAPINFOHEADER));

    // 获取错切参数
```

```
    getMirrorData();

    // 根据错切方向的选择和倍率修改图像参数，对“画布”的大小做适当的调整
    width = newImg->bmih.biWidth;
    height = newImg->bmih.biHeight;
    newImg->aBitmapBits
        = (BYTE *)malloc(sizeof(BYTE) * newImg->bmih.biSizeImage);

    // 错切图像
    for (y = 0; y < height; y++)
        for (x = 0; x < width; x++) {
            newPos = y * ColorBitWidth + x * 3;

            oldX = MirrorChoice ? width - x : x;
            oldY = !MirrorChoice ? height - y : y;

            // 如果计算出来的点不在原图范围内，则用白色像素替代
            if (oldX < 0 || oldX >= width
              || oldY < 0 || oldY >= height) {
                newImg->aBitmapBits[newPos] = 255;
                newImg->aBitmapBits[newPos + 1] = 255;
                newImg->aBitmapBits[newPos + 2] = 255;
                continue;
            }

            oldPos = oldY * ColorBitWidth + oldX * 3;
            newImg->aBitmapBits[newPos] = bf->aBitmapBits[oldPos];
            newImg->aBitmapBits[newPos + 1]
                                    = bf->aBitmapBits[oldPos + 1];
            newImg->aBitmapBits[newPos + 2]
                                    = bf->aBitmapBits[oldPos + 2];
        }

    return newImg;
}
```

镜像的具体实现过程为：

1. 拷贝旧图的文件头部分到新图
2. 获取镜像（用户可以自行设置，也可以直接使用默认值）
3. 由于镜像前后图像大小没有改变，因此无需调整画布大小
4. 通过最近邻插值来实现具体的镜像操作，以填补镜像带来的空洞

3.6 主程序 & 辅助函数

由于主程序和辅助函数结构较为简单，故这里仅列出代码，不作分析。

```
// 询问进行何种简单几何变换
int AskforChoicev4(void) {
    int choice;

    printf("Here are choices of simple geometric transform.\n");
    printf("10) Translation\n");
    printf("11) Rotation\n");
    printf("12) Scale\n");
    printf("13) Sheer\n");
    printf("14) Mirror\n");
    printf("Other numbers can cancel the transform.\n");
    printf("Please input your choice(10, 11, 12, 13 or 14): ");
    scanf("%d", &choice);
    if (choice < 10 || choice > 14) {
        printf("Simple geometric transform canceled!\n");
        exit(1);
    } else {
       printf("Valid choice!\nPlease wait a minute for the generation
of the new image...\n");
    }

    return choice;
}

#include "bmp.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
    int choice;
    BMPFILE oldImg, newImg;

    // 分配存储空间
    oldImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));
    newImg = (BMPFILE)malloc(sizeof(struct tagBMPFILESTRUCT));

    // 读取原图
    oldImg = ReadBMPFile();
```

```
    // 询问选择
    choice = AskforChoicev4();

    // 根据选择执行不同的简单几何变换
    switch (choice) {
        case 10: newImg = Translation(oldImg); break;
        case 11: newImg = Rotation(oldImg); break;
        case 12: newImg = Scale(oldImg); break;
        case 13: newImg = Sheer(oldImg); break;
        case 14: newImg = Mirror(oldImg); break;
        default: printf("Invalid choice!\n"); exit(1);
    }

    // 得到新图
    WriteBMPFile(newImg, choice);

    return 0;
}
```

四、 实验环境及运行方法

4.1 实验环境

我的开发环境如下:

- Windows OS 下:
  - Windows 11 24H2
  - gcc version 14.2.0
  - GNU Make 4.4.1
- Linux OS 下:
  - WSL 1
  - Ubuntu 24.04 LTS
  - gcc version 13.2.0
  - GNU Make 4.3

> **注意**
>
> 因为 Windows 下运行程序会有一个小 bug：在对 pepper.bmp（位于目录 `./代码/tests/pepper` 下）进行平移操作时，程序无法正确执行，经调试发现问题出在`WriteBMPFile()`函数中，但每次调试出问题的具体语句各不相同，相当奇怪；而 Ubuntu 则没有这个问题，所以我建议在 Linux 系统上执行程序。

4.2 运行方法

若要运行主程序，需要遵循以下步骤：

> **注意**
>
> 在执行以下命令前，请检查一下路径下 `./代码/test` 是否存在`.bmp` 图片，并且检查一下`./代码/scripts/bmp.h` 文件的宏定义 `BMPFILEPATH` 是否指的是该图片。若有问题请自行修改，否则程序无法正常运行。

1. 将目录切换至 `./代码/build`
2. 执行以下命令：

```
# 编译代码
$ make

# 运行可执行文件
$ ./lab2
Successfully open the file!
Size: 163154(bit)
ColorBitWidth: 700
Width: 233
Height: 233
Image Size: 163100
Here are choices of simple geometric transform.
10) Translation
11) Rotation
12) Scale
13) Sheer
14) Mirror
Other numbers can cancel the transform.
Please input your choice(10, 11, 12, 13 or 14):
```

现在程序询问执行何种简单几何变换，可以选择的方法有：

- 平移（序号 10）
- 旋转（序号 11）
- 缩放（序号 12）
- 错切（序号 13）
- 镜像（序号 14）

下面以输入 12 和 14 为例（它们包括了全部的三种问法，包括询问是否使用默认值，倍率和方向的确定）：

2.1） 输入 12

```
Please input your choice(10, 11, 12, 13 or 14): 12
Valid choice!
Please wait a minute for the generation of the new image...
Do you want to customize the scale data?
Please input yes or no:
```

现在程序询问用户是否由用户决定缩放倍率（yes 表示接受；no 表示拒绝，此时程序会使用默认值），这里以 yes 为例：

```
Please input yes or no:  yes
X:
```

接下来程序依次要求输入 x 方向和 y 方向的缩放倍率：

```
X:  1.5
Y:  1.5
Finish the conversion successfully!
```

此时程序完成了对图像的缩放，用户会得到一张宽高均为原图 1.5 倍的新图。

2.2） 输入 14

```
Please input your choice(10, 11, 12, 13 or 14): 14
Valid choice!
Please wait a minute for the generation of the new image...
Which mirror axis do you want to choose?
Please input x or y, otherwise the program will terminate!
Your choice:
```

现在程序询问用户按哪个方向实现镜像操作（x 表示沿 x 轴方向做镜像变换；y 表示沿 y 轴方向做镜像变换），这里以输入 y 为例：

```
Your choice:  y
Finish the conversion successfully!
```

此时程序完成了对图像的镜像变换。

3. 来到 ./代码/tests 目录，此时可以看到几何变换后的图像

五、 实验结果展示

> **注意**
>
> 本报告采用 typst 书写，但是 typst 不支持插入 BMP 图像文件，因此这里展示的是它们的 PNG 形式，对应的 BMP 形式见目录 `./代码/tests`。

先给出原图，以便于和后面几何变换后的图像做对比。由于报告的背景色是白色的，所以我给每张图片外边加了一个黑色边框，便于观察图像的变换（实际图片是没有这些边框的）。

Figure 11：原图

5.1 平移

Figure 12：图像沿 x 轴平移 50px，沿 y 轴平移 50px

Figure 13：图像沿 x 轴平移 100px，沿 y 轴平移 100px

Figure 14：图像沿 x 轴平移-50px，沿 y 轴平移-50px

Figure 15：图像沿 x 轴平移 700px，沿 y 轴平移 700px

虽然我们一般约定 x 轴和 y 轴方向分别是从左到右、从上到下，且原点位于图像左上角，但是由于 BMP 图像实际上是“倒着”存储在内存中，所以沿 y 方向的平移实际上是自底向上的。

5.2 旋转

Figure 16：逆时针旋转45°

Figure 17：逆时针旋转90°

Figure 18：逆时针旋转180°

5.3 缩放

Figure 19：缩放倍率 2×2

Figure 20：缩放倍率 0.5×0.5

Figure 20：缩放倍率 0.5×2

Figure 21：缩放倍率 2×0.5

5.4 错切

Figure 21：沿 x 方向错切，倍率为 2

Figure 22：沿 y 方向错切，倍率为 2

Figure 23：沿 x 方向错切，倍率为 1

5.5 镜像

Figure 24：沿 y 轴镜像

Figure 25：沿 x 轴镜像

综上，程序能够正确处理各种简单几何变换操作。

六、 心得体会

个人感觉本次作业的难点在于插值法的使用：虽然我选择了实现较为简单的最近邻插值，但在编写代码的过程中还是遇到了不少磕磕绊绊，比如“还原”几何变换后的像素点在原图的位置，稍有不慎就会导致计算错误，从而无法正确生成图像；对于超出原图范围的像素点，需要让它的颜色变成白色等。此外，虽然说是“简单”的几何变换，但是代码量还是不少的（~~可能是我代码设计的不够精巧吧~~）。

我在实现插值法之前，出于好奇，我先搞了一版没有用插值法的旋转操作，看看所谓的空洞问题，结果如下：

Figure 26：有“空洞”的图像

总而言之，我从本次实验中进一步理解了图像简单几何变换的原理，并掌握了相应的实现方法，使我受益颇多。