

# 浙江大学实验报告

专业：计算机科学与技术  
姓名：沈一芑  
学号：322010827  
日期：2023/11/27

课程名称：\_\_\_\_图像信息处理\_\_\_\_ 指导老师：\_\_\_\_宋明黎\_\_\_\_ 成绩：\_\_\_\_  
实验名称：\_\_\_\_图像的几何变换\_\_\_\_

## 1. 实验目的和要求

### 1.1. 实验目的

- 对图像进行几何变换

### 1.2. 实验要求

- 对图像进行平移变换 (Translation)
- 对图像进行旋转变换 (Rotation)
- 对图像进行镜像变换 (Mirror)
- 对图像进行错切变换 (Shear)
- 对图像进行缩放变换 (Scale)

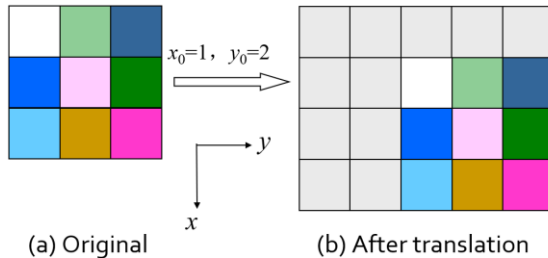
## 2. 实验内容和原理

### 2.1. 对图像进行平移变换 (Translation)

- Equation:

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

- Effect:



## 2.2. 对图像进行旋转变换 (Rotation)

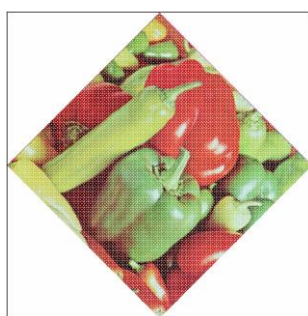
- Equation:

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

- Effect:



Origin



Rotation



Interpolaton

- Interpolation

在对图像直接进行旋转处理后, 由于坐标映射后为 double 型, 变换为像素点整型时进行取整, 使得新图像的部分区域没有映射。故需采取插值的方法补全空白空洞。

一种行插值的方法即将空洞赋值前一个像素点。另一种插值方法即将空洞进行反向坐标转换, 映射回原图像, 将原图像对应 RGB 赋值给新像素点。

## 2.3. 对图像进行镜像变换 (Mirror)

- Equation:

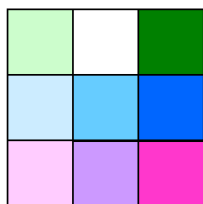
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

当  $S_x=1$  &&  $S_y=-1$ , 以 X 轴为对称轴镜像变换 (Vertical mirror)

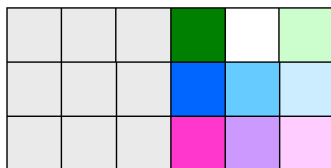
当  $S_x=-1$  &&  $S_y=1$ , 以 Y 轴为对称轴镜像变换 (Horizontal mirror)

\* 当  $S_x=-1$  &&  $S_y=-1$ , 按原点做对称镜像变换

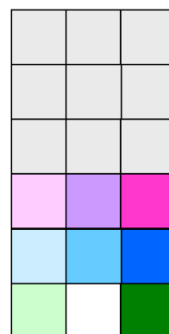
- Effect:



Origin



Horizontal



Vertical

## 2.4. 对图像进行错切变换 (Shear)

- Equation:

$$\begin{cases} a(x, y) = x + d_x y \\ b(x, y) = y \end{cases}$$

沿 X 轴错切

$$\begin{cases} a(x, y) = x \\ b(x, y) = y + d_y x \end{cases}$$

沿 Y 轴错切

也可 X, Y 轴同时错切

- Effect:



## 2.5. 对图像进行缩放变换 (Scale)

- Equation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} c & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Effect:



Origin

same ratio

diff ratio

diff ratio enlarge

当对图像进行放大时，同样会产生空洞，即新图像中由于放大无法在原图像中找到映射的像素点。同样应采取插值的方法进行空洞填补。但如此操作会产生放大比例大小的色块，即马赛克（mosaics）。

### 3. 实验步骤与分析

#### 3.1. 准备工作

- 定义结构体并声明变量如下；每个变量对应一张 BMP 图片：

```
• typedef struct{
•     BMFH bmfh;
•     BMIH bmih;
•     RGBQUAD rgbquad[256];
•     BYTE *data;
•     int LineBytes, WidthBMP, HeightBMP, SizeImageBMP;
•     //LineBytes is to compliment the width given that bytes mod 4 == 0
•     //size of image part, instead of whole bmp
• }IMAGE;
•
• IMAGE input, gray, translation, rotation, scale, shear, mirror;
```

- 根据原理中公式定义坐标转换函数（以及 rotation 的反向转换函数）。

```
• void coord_translation(Coord *initial_coord, int dx, int dy);
• void coord_rotation_back(Coord *initial_coord, double angle);
• void coord_rotation(Coord *initial_coord, double angle);
• void coord_mirror(Coord *initial_coord, int mx, int my);
• void coord_shear(Coord *initial_coord, double tx, double ty);
• void coord_scale(Coord *initial_coord, int cx, int cy);
```

- 编写 calc\_pos 函数，根据 LineBytes, x, y, resize\_x/y 映射定位位置。

```
• int calc_pos(Coord *initial_coord, int resize_x, int resize_y, int bits, int linebytes){  
•     int x = initial_coord->x;  
•     int y = initial_coord->y;  
•     x -= resize_x;  
•     y -= resize_y;  
•     return x * linebytes + y*bits;  
• }
```

- 编写 update\_boundary 函数，更新 vector 数组中存储的最大/最小边界。

```
• void update_boundary(Coord *new_coord, vector<int> *boundary){  
•     if(new_coord->x > boundary->at(2)) boundary->at(2)=new_coord->x;  
•     if(new_coord->x < boundary->at(0)) boundary->at(0)=new_coord->x;  
•     if(new_coord->y > boundary->at(3)) boundary->at(3)=new_coord->y;  
•     if(new_coord->y < boundary->at(1)) boundary->at(1)=new_coord->y;  
• }
```

- 编写 resize 函数，根据 OPERATION\_TYPE 枚举变量确定操作类型，调用相应坐标转换公式，更新图像四个顶点的坐标，与初始值比较，取最大差值作为新图像的 Height 与 Width 并以此计算 LineBytes(末尾 4 位补齐)和图像大小信息。最后更新 resize\_x/y 以将像素点映射至[0, new\_Height/Width]。

```
• void resize(IMAGE *initial, vector<double> *parameter, int *resize_x, int *resize_y){  
•  
•     vector<int> boundary={0,0,initial->bmih.biHeight,initial->bmih.biWidth};  
•     Coord new_vertices[4]={0,0}, {0, boundary[3]}, {boundary[2], 0},  
•     {boundary[2], boundary[3]};  
•  
•     switch(OPERATION_TYPE){  
•         case TRANSLATION:  
•             for(int i=0;i<4;i++)  
•                 coord_translation(&new_vertices[i], parameter->at(0),  
•                 parameter->at(1));  
•             break;  
•         case ROTATION:
```

```

•         for(int i=0;i<4;i++)
•             coord_rotation(&new_vertices[i], parameter->at(0));
•         break;
•         case MIRROR:
•             for(int i=0;i<4;i++)
•                 coord_mirror(&new_vertices[i], parameter->at(0),
parameter->at(1));
•             break;
•         case SHEAR:
•             for(int i=0;i<4;i++)
•                 coord_shear(&new_vertices[i], parameter->at(0),
parameter->at(1));
•             break;
•         case SCALE:
•             for(int i=0;i<4;i++)
•                 coord_scale(&new_vertices[i], parameter->at(0),
parameter->at(1));
•             break;
•         default:
•             cout << "something went wrong in operation type." << endl; break;
•     }
•
•     for(int i=0;i<4;i++)    update_boundary(&new_vertices[i], &boundary);
•
•     initial->HeightBMP = initial->bmih.biHeight = boundary[2]-boundary[0];
•     initial->WidthBMP = initial->bmih.biWidth = boundary[3]-boundary[1];
•     initial->LineBytes = (initial->WidthBMP * (initial->bmih.biBitCount / 8)
+ 3) / 4 * 4;
•     initial->SizeImageBMP = initial->bmih.biSizeImage = initial->HeightBMP *
initial->LineBytes;
•     initial->bmfh.bfSize = initial->bmfh.bfOffBits + initial->SizeImageBMP;
•
•     *resize_x = boundary[0];
•     *resize_y = boundary[1];
•
•     return;
•
• }

```

### 3.2. 平移 Translation

- 复制 input 图像 BMFH, BMIH 信息至 translation。
- 将参数 dx, dy 插入 vector 数组, 更改操作状态为平移。
- 调用 resize 函数, 根据平移状态调用相应函数 coord\_translation 计算新图像信息。
- 根据更新后的图像大小信息为 data 指针分配空间, 并赋初值 255 (白色)。
- 遍历原图像像素点, 将坐标通过公式进行转换, 为新图像赋值。

```
• void do_translation(IMAGE *input, IMAGE *output, int dx, int dy){  
•  
•     memcpy(&(output->bmfh), &(input->bmfh), sizeof(BMFH));  
•     memcpy(&(output->bmih), &(input->bmih), sizeof(BMIH));  
•     vector<double> parameter;  
•     parameter.push_back(dx);  
•     parameter.push_back(dy);  
•     OPERATION_TYPE=TRANSLATION;  
•     int resize_x, resize_y;  
•     resize(output, &parameter, &resize_x, &resize_y);  
•  
•     output->data = new BYTE[output->SizeImageBMP];  
•     for(int i=0;i<output->bmih.biHeight;i++){  
•         for(int j=0;j<output->LineBytes;j++){  
•             output->data[i*output->LineBytes+j]=255;  
•         }  
•     }  
•     for(int i=0;i<input->HeightBMP;i++){  
•         for(int j=0;j<input->WidthBMP;j++){  
•             Coord now={i, j};  
•             int in_pos=calc_pos(&now, 0, 0, input->bmih.biBitCount/8,  
input->LineBytes);  
•             coord_translation(&now, dx, dy);  
•             int out_pos=calc_pos(&now, resize_x, resize_y,  
output->bmih.biBitCount/8, output->LineBytes);  
•             output->data[out_pos]=input->data[in_pos];  
•             output->data[out_pos+1]=input->data[in_pos+1];  
•             output->data[out_pos+2]=input->data[in_pos+2];  
•         }  
•     }  
• }
```

### 3.3. 旋转 Rotation

- 复制 input 图像 BMFH, BMIH 信息至 rotation。
- 将参数 angle 插入 vector 数组，更改操作状态为旋转。
- 调用 resize 函数，根据平移状态调用相应函数 coord\_rotation 计算新图像信息。
- 根据更新后的图像大小信息为 data 指针分配空间，并赋初值 255（白色）。
- 遍历原图像像素点，将坐标通过公式进行转换，为新图像赋值。
- 遍历新图像空洞，用反向转换公式映射回原图；若在范围内，则用原图像素点插值。

```
• void do_rotation(IMAGE *input, IMAGE *output, double angle){
•
•     memcpy(&(output->bmfh), &(input->bmfh), sizeof(BMFH));
•     memcpy(&(output->bmih), &(input->bmih), sizeof(BMIH));
•
•     vector<double> parameter;
•     parameter.push_back(angle);
•
•     OPERATION_TYPE=ROTATION;
•
•     int resize_x, resize_y;
•     resize(output, &parameter, &resize_x, &resize_y);
•
•     output->data = new BYTE[output->SizeImageBMP];
•     for(int i=0;i<output->bmih.biHeight;i++)
•         for(int j=0;j<output->LineBytes;j++)
•             output->data[i*output->LineBytes+j]=255;
•
•     for(int i=0;i<input->HeightBMP;i++){
•         for(int j=0;j<input->WidthBMP;j++){
•             Coord now={i, j};
•             int in_pos=calc_pos(&now, 0, 0, input->bmih.biBitCount/8,
input->LineBytes);
•             coord_rotation(&now, angle);
•             int out_pos=calc_pos(&now, resize_x, resize_y,
output->bmih.biBitCount/8, output->LineBytes);
•             output->data[out_pos]=input->data[in_pos];
•             output->data[out_pos+1]=input->data[in_pos+1];
•             output->data[out_pos+2]=input->data[in_pos+2];
•         }
•     }
• }
```



```

•     for(int i=0;i<output->HeightBMP;i++){
•         for(int j=0;j<output->WidthBMP;j++){
•             Coord back={i, j};
•             int out_pos=calc_pos(&back, 0, 0, output->bmih.biBitCount/8,
output->LineBytes);
•             back.x+=resize_x;
•             back.y+=resize_y;
•
•             coord_rotation_back(&back, angle);
•
•             if(back.x<0||back.y<0||back.x>=input->HeightBMP||back.y>=input->W
idthBMP)
•                 continue;
•             int in_pos=calc_pos(&back, 0, 0, input->bmih.biBitCount/8,
input->LineBytes);
•
•             output->data[out_pos]=input->data[in_pos];
•             output->data[out_pos+1]=input->data[in_pos+1];
•             output->data[out_pos+2]=input->data[in_pos+2];
•
•         }
•     }
• }

```

### 3.4. 镜像 Mirror

- 复制 input 图像 BMFH, BMIH 信息至 mirror。
- 将参数 mx, my 插入 vector 数组，更改操作状态为镜像。
- 调用 resize 函数，根据平移状态调用相应函数 coord\_mirror 计算新图像信息。
- 根据更新后的图像大小信息为 data 指针分配空间，并赋初值 255（白色）。
- 遍历原图像像素点，将坐标通过公式进行转换，为新图像赋值。

```

• void do_mirror(IMAGE *input, IMAGE *output, int mx, int my){
•
•     memcpy(&(output->bmfh), &(input->bmfh), sizeof(BMFH));
•     memcpy(&(output->bmih), &(input->bmih), sizeof(BMIH));
•
•     vector<double> parameter;
•     parameter.push_back(mx);
•     parameter.push_back(my);

```

```

• OPERATION_TYPE=MIRROR;
• int resize_x, resize_y;
• resize(output, &parameter, &resize_x, &resize_y);
• output->data = new BYTE[output->SizeImageBMP];
•
• for(int i=0;i<output->bmih.biHeight;i++){
•     for(int j=0;j<output->LineBytes;j++){
•         output->data[i*output->LineBytes+j]=255;
•     }
• }
• for(int i=0;i<input->HeightBMP;i++){
•     for(int j=0;j<input->WidthBMP;j++){
•         Coord now={i, j};
•         int in_pos=calc_pos(&now, 0, 0, input->bmih.biBitCount/8,
input->LineBytes);
•         coord_mirror(&now, mx, my);
•         int out_pos=calc_pos(&now, resize_x, resize_y,
output->bmih.biBitCount/8, output->LineBytes);
•         output->data[out_pos]=input->data[in_pos];
•         output->data[out_pos+1]=input->data[in_pos+1];
•         output->data[out_pos+2]=input->data[in_pos+2];
•     }
• }
• }

```

### 3.5. 放缩 Scale

- 复制 input 图像 BMFH, BMIH 信息至 scale。
- 将参数 cx, cy 插入 vector 数组，更改操作状态为放缩。
- 调用 resize 函数，根据平移状态调用相应函数 coord\_scale 计算新图像信息。
- 根据更新后的图像大小信息为 data 指针分配空间，并赋初值 255（白色）。
- 遍历新图像像素点，将坐标反向转换，用原图像像素点赋值。

```

•
• void do_scale(IMAGE *input, IMAGE *output, double cx, double cy){
•
•     memcpy(&(output->bmfh), &(input->bmfh), sizeof(BMFH));
•     memcpy(&(output->bmih), &(input->bmih), sizeof(BMIH));
•
•     vector<double> parameter;
•     parameter.push_back(cx);
•     parameter.push_back(cy);

```

```

•
• OPERATION_TYPE=SCALE;
• int resize_x, resize_y;
• resize(output, &parameter, &resize_x, &resize_y);
•
• output->data = new BYTE[output->SizeImageBMP];
• for(int i=0;i<output->bmih.biHeight;i++)
•     for(int j=0;j<output->LineBytes;j++)
•         output->data[i*output->LineBytes+j]=255;
•
• for(int i=0;i<output->HeightBMP;i++){
•     for(int j=0;j<output->WidthBMP;j++){
•
•         Coord now={i, j};
•         int in_pos=calc_pos(&now, 0, 0, output->bmih.biBitCount/8,
output->LineBytes);
•         // coord_scale(&now, cx, cy);
•         double ix=now.x, iy=now.y;
•         ix/=cx, iy/=cy;
•         now.x=(int)ix, now.y=(int)iy;
•
•         if(now.x<0||now.y<0||now.x>=input->HeightBMP||now.y>=input->Width
BMP) continue;
•         int out_pos=calc_pos(&now, 0, 0, input->bmih.biBitCount/8,
input->LineBytes);
•         output->data[in_pos]=input->data[out_pos];
•         output->data[in_pos+1]=input->data[out_pos+1];
•         output->data[in_pos+2]=input->data[out_pos+2];
•     }
• }
• }

```

### 3.6. 错切 Shear

- 复制 input 图像 BMFH, BMIH 信息至 shear。
- 将参数插入 vector 数组，更改操作状态为错切。
- 调用 resize 函数，根据平移状态调用相应函数 coord\_shear 计算新图像信息。
- 根据更新后的图像大小信息为 data 指针分配空间，并赋初值 255（白色）。
- 遍历原图像像素点，将坐标通过公式进行转换，为新图像赋值。






```

• void do_shear(IMAGE *input, IMAGE *output, double dx, double dy){
•
•     memcpy(&(output->bmfh), &(input->bmfh), sizeof(BMFH));
•     memcpy(&(output->bmih), &(input->bmih), sizeof(BMIH));
•
•     vector<double> parameter;
•     parameter.push_back(dx);
•     parameter.push_back(dy);
•
•     OPERATION_TYPE=SHEAR;
•     int resize_x, resize_y;
•     resize(output, &parameter, &resize_x, &resize_y);
•
•     output->data = new BYTE[output->SizeImageBMP];
•     for(int i=0;i<output->bmih.biHeight;i++){
•         for(int j=0;j<output->LineBytes;j++){
•             output->data[i*output->LineBytes+j]=255;
•
•         }
•     }
•     for(int i=0;i<input->HeightBMP;i++){
•         for(int j=0;j<input->WidthBMP;j++){
•             Coord now={i, j};
•             int in_pos=calc_pos(&now, 0, 0, input->bmih.biBitCount/8,
input->LineBytes);
•             coord_shear(&now, dx, dy);
•             int out_pos=calc_pos(&now, resize_x, resize_y,
output->bmih.biBitCount/8, output->LineBytes);
•             output->data[out_pos]=input->data[in_pos];
•             output->data[out_pos+1]=input->data[in_pos+1];
•             output->data[out_pos+2]=input->data[in_pos+2];
•
•         }
•     }
• }




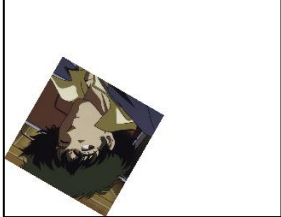




```

4. 实验结果展示


4.1. 平移 Translation

Origin	dx=100 dy=100	dx=-100 dy=100	dx=100 dy=-100	dx=-100 dy=-100
				








4.2. 旋转 Rotation

origin	angle=30	angle=120	angle=210
			
angle=90	angle=180		angle=360
			





### 4.3. 镜像 Mirror

origin	$mx=1, my=-1$	$mx=-1, my=1$	$mx=-1, my=-1$
			

### 4.4. 缩放 Scale

origin	$cx=2, cy=2$		$cx=0.5, cy=0.5$	
		图像 分辨率 1920 x 1920 宽度 1920 像素 高度 1920 像素 位深度 24		图像 分辨率 480 x 480 宽度 480 像素 高度 480 像素 位深度 24
origin	$cx=1, cy=2$	$cx=2, cy=1$	$cx=1, cy=0.5$	$cx=0.5, cy=1$
图像 分辨率 960 x 960 宽度 960 像素 高度 960 像素 位深度 24				

### 4.5. 错切 Shear

origin	$tx=30, ty=0$	$tx=0, ty=30$	$tx=30, ty=30$
			

## 5. 心得体会

通过这次实验我了解了图像的几何变换，并亲自实现平移、旋转、镜像、错切、缩放。

本次实验的主要难点是处理图像几何变换后画幅大小的变动。同时，在进行本次实验时尽量加入模块化设计的思路，将不同变换相同的操作部分统一设计。在计算图幅大小时，选取了图像的四个顶点，计算变换后顶点的位置，进而做差更新图像的大小。

此处有一个要注意的点：为了展示图像的几何变换过程，在更新图像大小时，Height 和 Width 的更新是根据几何变换后的边界顶点位置和几何变换前的边界顶点位置共同决定的。故而如果想仅仅查看几何变换后的图像，则将 `resize` 部分的初始边界全部赋值为 0。

而本次实验另一个重要的步骤即插值填补空洞。可通过行插值、双线性插值等方法完善图像。但此处有一个问题：前向映射和后向映射插值两种方法，是否可以理解为前者是对原图像完全地映射变化，而后者已经是在原图像基础上进行了部分优化？

综上，希望能够在以后的实验中用到本次所学的几何变换预处理矫正图片，为后续提供便利。

2023. 11. 27