1. 概述

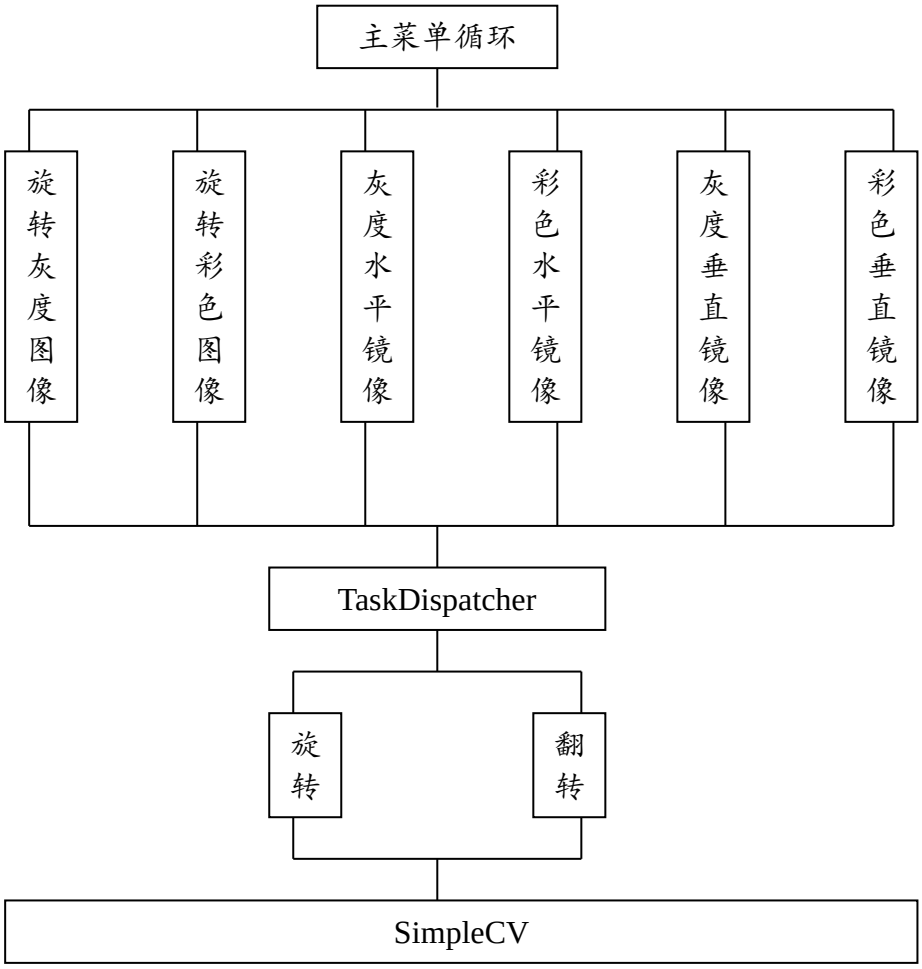
1.1 问题分析

数字图像处理，实现图片的灰度化以及一系列几何变换，并写入到 bmp 文件。

1.2 功能

- ✓ 读取彩色 bmp 图片；
- ✓ 将彩色图像转换成灰度图像；
- ✓ 将彩色或灰度图像旋转任意角度；
- ✓ 将彩色或灰度图像水平或垂直翻转；
- ✓ 将变换后的图像保存到新的 bmp 文件。

2. 总体逻辑结构


主菜单循环
旋转灰度图像
旋转彩色图像
灰度水平镜像
彩色水平镜像
灰度垂直镜像
彩色垂直镜像
TaskDispatcher
旋转
翻转
SimpleCV

3. 详细实现

具体的实现分为两部分。首先最重要的部分——SimpleCV 库，也就是源码 simplecv 目录下的代码，这部分主要实现通用的、类似于 OpenCV 的图像操作，与选题本身并无耦合，可用于任务书中任意一个图像处理的题，接口参考 OpenCV，但有一些细微差别，在本次课程设计的语境下，使用起来更简单；第二部分，选题的实现，实现菜单选项的打印、根据用户输入实现所要求的功能，也就是 main.cpp 文件中的代码。（代码本身用纯 C 编写，源文件格式为 cpp 只是为了方便在 VC6.0 中编译。）

3.1 SimpleCV

SimpleCV 库主要由四部分组成：types、core、matrix、io，分别用于定义类型、实现主要图像处理函数、矩阵运算、输入输出图像文件。

3.1.1 types

也即 types.h，该文件中主要定义了一些需要暴露给库的使用者的数据类型，并对结构体提供简单的构造方法(在栈上分配内存),这些类型包括 ScvPoint、ScvSize、ScvMat、ScvPixel、ScvImage、ScvHistogram，分别用来表示点、大小、矩阵、像素、图像、灰度直方图。

3.1.2 core

包括 core.h、core.cpp 两个文件，h 文件里是对库的使用者暴露的接口声明，cpp 文件里是接口的实现。这一部分是 SimpleCV 库的核心，首先实现 ScvImage、ScvMat、ScvHistogram 的创建、复制、释放操作，然后实现了灰度直方图计算、仿射变换（采用最邻近插值法）、灰度化（提供 RGB 分量、最大值、平均值、加权平均值这几种灰度计算方法）、二值化、分离 RGB、反色、直方图均衡化、平滑去噪（提供中值、均值、高斯这几种平滑算法）、加权图像叠加、Canny 算法边缘检测。另外，提供方便地获取和设置某坐标上的像素的函数，分别是 scvGetPixel()、scvSetPixel()，从而方便库的使用者实现自己的图像处理操作。

3.1.3 matrix

包括 matrix.h、matrix.cpp，这一部分主要实现一些基本的矩阵运算，如点乘、数乘、求逆、求行列式、求伴随矩阵、求转置。因为 core 部分的仿射变换函数，需要计算逆矩阵，而计算逆矩阵需要进行其它矩阵运算，为了避免 core 部分过于臃肿，将它们分离到 matrix 部分实现，并最大限度降低耦合，如果以后在其它地方需要用到矩阵运算，也可以单独调用这一部分。

3.1.4 io

包括 io.h、io.cpp，这一部分主要实现图片文件的读取和写入，当然现在只支持 24 位 bmp 文件。bmp 文件的读写就是分别读写文件头、信息头、图像数据，由于部分图片是从下往上扫描的，因此也在这里辨别了图片的方向，并保存在 ScvImage 中，以便后面的图像处理。

3.2 主函数

主函数里首先读取参数里指定的图片文件，然后进入主菜单循环，输出选项菜单让用户选择。这里接受用户选择之后，采用了任务调度模型，将用户所选择的操作要求封装到一个 TaskMassage 结构体，并统一发送到 dispatchTask()函数，之后该函数再根据任务类型分配给不同的函数来实现操作。任务调度模型的采用显著减少了函数数量和冗余代码，并且将控制台输入输出的逻辑和真正图像处理的逻辑分开，降低偶合，也使后续的功能扩展更加方便。任务调度器的函数代码如下：

```
void dispatchTask(TaskMessage msg) {
    ScvImage *image;
    if (msg.what & TASK_FLAG_GRAYING) {
        if (gGrayImage == NULL) {
            gGrayImage = scvCreateImage(scvGetSize(gSrcImage));
            scvGraying(gSrcImage, gGrayImage, SCV_GRAYING_W_AVG);
        }
        image = scvCloneImage(gGrayImage);
        msg.what &= ~TASK_FLAG_GRAYING;
    } else {
        image = scvCloneImage(gSrcImage);
    }
    switch (msg.what) {
        case TASK_ROTATE:
            rotateImage(image, msg.val.floatVal);
            break;
        case TASK_FLIP:
            flipImage(image, (SCV_FLIP_TYPE) msg.val.intVal);
            break;
        default:
            break;
    }
    saveImage(image, gOutFilePath);
    scvReleaseImage(image);
}
```

首先判断是否有灰度化这个 flag，如果有则先灰度化，然后将旋转和翻转操作分别分配给 rotateImage()和 flipImage()来执行。

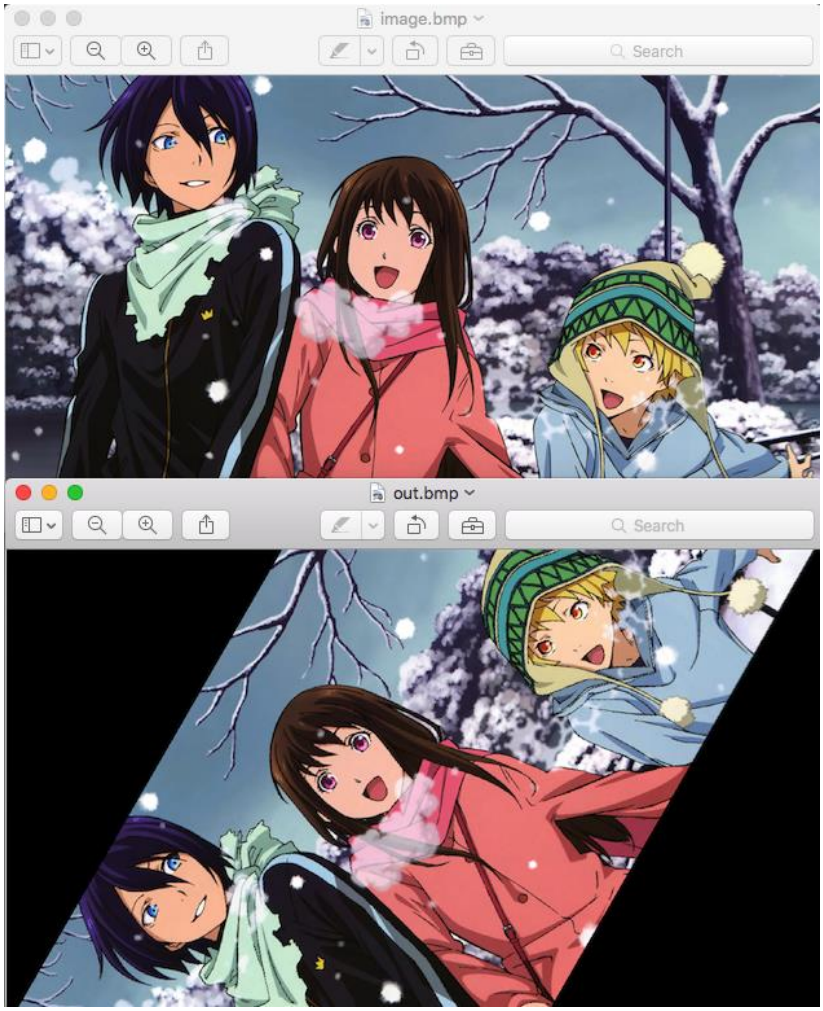
4. 运行示例

```
build ➤ ./6_scv ../image.bmp -o ../out.bmp


---------数字图像处理 几何变换1 v1.0.0---------

0. 退出
1. 旋转灰度图像
2. 旋转彩色图像
3. 灰度水平镜像
4. 彩色水平镜像
5. 灰度垂直镜像
6. 彩色垂直镜像

请输入选项：2
请输入要旋转的角度：60
操作成功！变换后的图像已经输出到 ../out.bmp.
```



5. 使用说明

在命令行使用下面命令运行：
./6_scv ../image.bmp -o ../out.bmp
其中第二个参数指定原图片的路径，第三个参数是“-o”，第四个参数指定输出图片的路径。

6. 源码

```
// main.cpp

#include <stdio.h>
#include <string.h>
#include "simplecv/scv.h"

char *gOutFilePath;
char *gSrcFilePath;
ScvImage *gSrcImage;
ScvImage *gGrayImage;

const int TASK_FLAG_GRAYING = 0x10;
const int TASK_ROTATE = 0x01;
const int TASK_FLIP = 0x02;

typedef struct _TaskMessage {
    int what;
    union {
        int intVal;
        float floatVal;
        void *obj;
    } val;
} TaskMessage;

void dispatchTask(TaskMessage msg);

int main(int argc, char **argv) {
    printf("\n\n---------数字图像处理 几何变换 1 v1.0.0---------\n\n");

    if (argc == 1) {
        printf("  用  法  :  command-name  <source-file-path>  -o
```

```
<output-file-path>\n");
        return 0;
    } else if (0 == strcmp("-o", argv[2]) && 4 == argc) {
        gSrcFilePath = argv[1];
        gOutFilePath = argv[3];
    } else {
        printf("参数有误.\n");
        return 1;
    }

    gSrcImage = scvLoadImage(gSrcFilePath);

    int choice;
    // Main loop
    for (; ;) {
        printf("0. 退出\n"
                    "1. 旋转灰度图像\n"
                    "2. 旋转彩色图像\n"
                    "3. 灰度水平镜像\n"
                    "4. 彩色水平镜像\n"
                    "5. 灰度垂直镜像\n"
                    "6. 彩色垂直镜像\n");
        printf("\n 请输入选项: ");
        scanf("%d", &choice);
        TaskMessage msg = {0};
        switch (choice) {
            case 0:
                return 0;
            case 1:
                msg.what |= TASK_FLAG_GRAYING;
            case 2:
                msg.what |= TASK_ROTATE;
                printf("请输入要旋转的角度: ");
                scanf("%f", &msg.val.floatVal);
                dispatchTask(msg);
                printf("操作成功! 变换后的图像已经输出到 %s.\n",
gOutFilePath);
                break;
            case 3:
                msg.what |= TASK_FLAG_GRAYING;
            case 4:
                msg.what |= TASK_FLIP;
                msg.val.intVal = SCV_FLIP_HORIZONTAL;
                dispatchTask(msg);
```

```
                printf("操作成功！变换后的图像已经输出到 %s.\n",
gOutFilePath);
                break;
            case 5:
                msg.what |= TASK_FLAG_GRAYING;
            case 6:
                msg.what |= TASK_FLIP;
                msg.val.intVal = SCV_FLIP_VERTICAL;
                dispatchTask(msg);
                printf("操作成功！变换后的图像已经输出到 %s.\n",
gOutFilePath);
                break;
            default:
                // Prompt again.
                break;
        }
        printf("\n");
    }
}

void saveImage(ScvImage *image, const char *path) {
    scvSaveImage(image, path);
}

void rotateImage(ScvImage *src, float angle) {
    ScvMat *mat = scvCreateMat(2, 3);
    scvRotationMatrix(scvGetCenter(src), angle, mat);
    scvWarpAffine(src, src, mat, scvPixelAll(0));
}

void flipImage(ScvImage *src, SCV_FLIP_TYPE type) {
    ScvMat *mat = scvCreateMat(2, 3);
    scvFlipMatrix(scvGetCenter(src), type, mat);
    scvWarpAffine(src, src, mat, scvPixelAll(0));
}

void dispatchTask(TaskMessage msg) {
    ScvImage *image;

    if (msg.what & TASK_FLAG_GRAYING) {
        if (gGrayImage == NULL) {
            gGrayImage = scvCreateImage(scvGetSize(gSrcImage));
            scvGraying(gSrcImage, gGrayImage, SCV_GRAYING_W_AVG);
        }
```

```
        image = scvCloneImage(gGrayImage);
        msg.what &= ~TASK_FLAG_GRAYING;
    } else {
        image = scvCloneImage(gSrcImage);
    }

    switch (msg.what) {
        case TASK_ROTATE:
            rotateImage(image, msg.val.floatVal);
            break;
        case TASK_FLIP:
            flipImage(image, (SCV_FLIP_TYPE) msg.val.intVal);
            break;
        default:
            break;
    }

    saveImage(image, gOutFilePath);
    scvReleaseImage(image);
}
```

7. 心得体会

做图像处理的话，感觉数学要求还是挺高的，线性代数、概率统计等都会用到，写 SimpleCV 的时候，在仿射变换那边翻了好久线性代数的书，然后直方图拉伸和边缘检测有很多概率的东西，不过这个毕竟没学过，只能去网上找了。

另外这次的 SimpleCV 也是第一次真正拿 C 写一些比较有用的东西，在代码逻辑、模块划分上面有所进步，写代码更加有逻辑性。

8. 实习日志

- 6 月 19 日，基于 OpenCV 2.4.13 实现几何变换功能。
- 6 月 20 日，实现存储图像到 bmp 文件。
- 6 月 21 日~22 日，发现 OpenCV 2.x 不支持 VC6.0，并且 OpenCV 1.0 装起来太麻烦，于是自己写了一个 SimpleCV。
- 6 月 25 日，基于 SimpleCV 实现几何变换题目。

9. 参考文献

1. http://crazycat1130.pixnet.net/blog/post/1345538，點陣圖（Bitmap）檔案格式；
2. http://www.51itong.net/c-bmp-4592.html，c++创建 BMP 文件并写入数据；
3. http://en.wikipedia.org/wiki/Otsu%27s_method，Wikipedia: Otsu's method；
4. http://en.wikipedia.org/wiki/Histogram_equalization,Wikipedia:Histogram equalization；
5. http://www.kancloud.cn/digest/imageproebow/122463，高斯平滑滤波的 C++实现；
6. http://blog.csdn.net/likezhaobin/article/details/6892629，Canny 边缘检测算法原理及其 VC 实现详解。