开发环境说明

> 操作系统：windows11
> 编译环境：SEGGER Embedded Studio for ARM v6.34a(64-bit)
> ARM架构：v4T
> ARM Core Type：ARM9

思路分析

因为ARM9嵌入式使用C语言编程，所以先在不使用各种预编译的图像处理的库函数的前提下，通过C语言编程实现图像旋转，再通过SEGGER仿真，测试程序性能。
ARM9可以直接读入RAW格式的图片，所以可以生成128x128（8bit）的RAW格式的灰度图像作为范例进行处理。

图像处理

方法一

windows系统中的画图工具可以直接修改图片的大小以及格式，将其保存为8bit的BMP格式，之后再通过C语言程序将其转化为RAW格式。

方法二

可以直接通过PHOTOSHOP软件更改图片颜色信息、图片格式和图片像素大小，直接生成128x128的RAW格式灰度图像（8bit）。

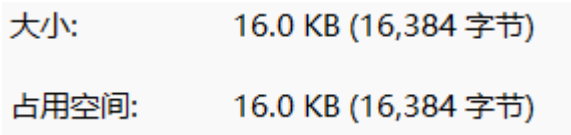
图像旋转算法

通过查阅资料，发现可以通过直接插值，最邻近插值，双线性内插值，以及双立方（三次）卷积法等实现图像旋转，而其中双线性插值计算有着较低的复杂度和较高的照片质量，对于一款性能有限的ARM9芯片来说，使用双线性插值计算的性价比较高.

基本原理

图片旋转原理

首先，RAW图片的信息都以像素值的方式存储在每个像素点上，而128x128的8bit灰度图像，在计算机中存储方式是一个128x128的数组，其中每个数据的大小为一字节。



将一幅图片旋转45度相当于将数组中的数据旋转45度，下图使用5x5的数组举例

35	57	74	91	72
39	17	14	90	22
21	21	23	15	86
44	33	68	17	64
60	29	38	49	25

若将其旋转45度之后会形成下图所示的数组，其中蓝色的0表示新形成的空位。易得矩阵围绕中心点旋转，将矩阵旋转会形成新的空白区域扩大数组大小，扩大的大小可以用几何方法求解。

0	0	0	0	35	0	0	0	0
0	0	0	39	0	57	0	0	0
0	0	21	0	17	0	74	0	0
0	44	0	21	0	14	0	91	0
60	0	33	0	23	0	90	0	72
0	29	0	68	0	15	0	22	0
0	0	38	0	17	0	86	0	0
0	0	0	49	0	64	0	0	0
0	0	0	0	25	0	0	0	0

设height为图片高度，width为图片宽度，则新生成的图片边长为 | height x cos(θ) | + | width x sin(θ) | 向上取整。
此时，设原图像坐标为(Xp, Yp)，中心点坐标为(cenX_p, cenY_p)新图像高度为heightf，宽度为widthf，像素坐标为(Xf, Yf)则原图像像素坐标和新图像像素坐标的对应关系为：

> xp = cos(θ) x Xf - sin(θ) x Yf + cenX_p;
> yp = cos(θ) x Xf + sin(θ) x Yf + cenX_p;

这样就会导致会计算出来的点并非实数，若直接用这种方式取整计算，会使得图片质量下降。

双线性插值原理

若计算新图像中的(1,0)点对应原图像的坐标会得到(65.14, 65.14)
双线性插值则是取计算结果附近的四个点，通过加权的方式将四个点的像素组合起来，得到当前位置点的像素值。
如上例可以表示为(65+0.14, 65+0.14)，即(i+u, j+v)，
i，j为坐标值整数部分；u，v为坐标值小数部分，
计算公式可以表示为：f(i+u,j+v) = (1-u)(1-v)f(i,j) + (1-u)vf(i,j+1) + u(1-v)f(i+1,j) + uvf(i+1,j+1)
以点之间的横纵坐标的乘积距离作为权重计算当前点的像素值。

性能优化

上述公式在计算过程中会涉及到较多的浮点数乘法运算，浮点数乘法运算相较于整数乘法会更为复杂，若将浮点数运算转化为整数运算，计算速度会有所提升。

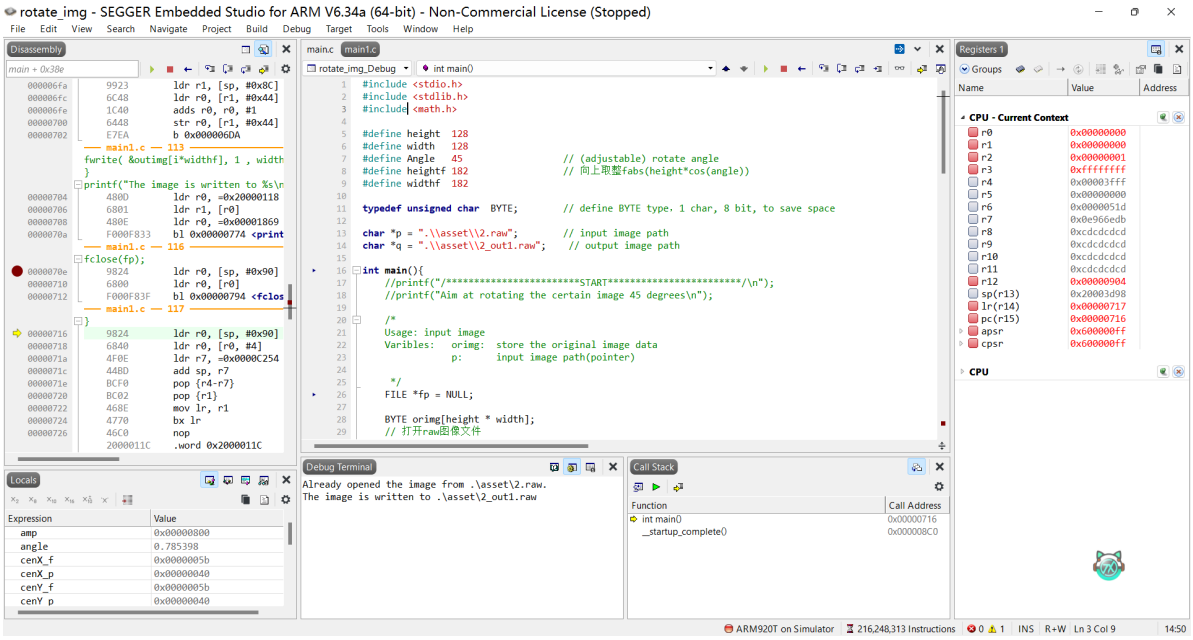
```
/*
将浮点数扩大amp倍，化成整数，减少浮点数的运算
 */
int u = (fXp - Xp)*amp; //浮点坐标小数部分，左移cout位
int v = (fYp - Yp)*amp;
//printf("%d %d\n", u, v);
```

```
/*
f(i+u,j+v) = (1-u)(1-v)f(i,j) + (1-u)vf(i,j+1) + u(1-v)f(i+1,j) + uvf(i+1,j+1)
再右移7+7=14位，还原原来扩大的倍数
 */
if(Xp >= 0 && Xp < height && Yp >= 0 && Yp < width) {//在原图范围内
  outimg[i * widthf + j] = ( (amp-u)*(amp-v)*   orimg[Xp * width + Yp] +
                             (amp-u)*v*  orimg[Xp * width + Yp+(Yp == width - 1
? 0 : 1 )] +
                             u*(amp-v)*  orimg[(Xp+(Xp == height - 1 ? 0 : 1 ))
* width +
                             Yp] +
                             u*v*          orimg[(Xp+(Xp == height - 1 ? 0 : 1 ))
* width +
                             Yp+(Yp == width - 1 ? 0 : 1 )] ) >> (cout*2);
```

先将浮点值u，v扩大amp倍，amp = 2^cout，在计算完成后再将计算结果右移cout x 2位，可以一定程度上加快程序性能。

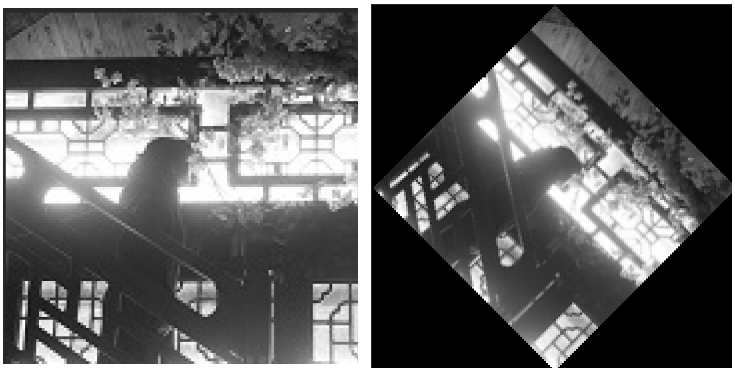
程序仿真

在SEGGER上面仿真：
在SEGGER工程中新建文件夹存放需要读入的图片，与KEIL不同的是，SEGGER可以自动通过程序读入图片并实现输出。



仿真结果

使用的图片和旋转后的图片对比：



空余部分程序缺省为0，像素表现为黑色。

仿真结果分析

对于一款给定的芯片，在编写类似的程序时，若想提升程序的运行性能，有时可以从程序编写、空间占用以及算法运算等软件层面入手，改善程序的运行速度。

程序源码

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include<time.h>

#define height  128
#define width   128
#define Angle   45                  // (adjustable) rotate angle
#define heightf 182                 // 向上取整fabs(height*cos(angle))
#define widthf  182

typedef unsigned char  BYTE;        // define BYTE type, 1 char, 8 bit, to save
space

char *p = ".\\asset\\2.raw";        // input image path
char *q = ".\\asset\\2_out1.raw";    // output image path

int main(){

    clock_t start,finish; // 定义变量
    double time;            //
    start = clock();
    //printf("/************************START************************/\n");
    //printf("Aim at rotating the certain image 45 degrees\n");

    /*
    Usage: input image
    Varibles:   orimg:  store the original image data
                p:      input image path(pointer)

     */
    FILE *fp = NULL;

    BYTE orimg[height * width];
    // 打开raw图像文件
    if((fp = fopen( p, "rb" )) == NULL) {
        printf("can not open the image\n " );
        return 0;
    } else {
        printf("Already opened the image from %s.\n", p);
    }

    int i, j;
    for( i = 0; i < height; i++ ) {
        // 逐行读入图像数据
        fread( &orimg[i*width], 1 , width, fp );
    }
    fclose(fp);
```

```
    /*
    Usage:  rotate image
    Note:   采用双线性插值法

     */
    //printf("Start rotating.\n");

    int cenX_p, cenY_p, cenX_f, cenY_f;      //旋转前后的中心点的坐标
    cenX_p = width / 2;                      // 64
    cenY_p = height / 2;                     // 64

    int     Xp, Yp, Xf, Yf;                  //旋转前后对应的像素点整数坐标
    double  fXp, fYp;                        //对应的浮点坐标（计算所得浮点数坐标）
    double  angle = (double)1.0 * Angle * 3.1415926 / 180;
    cenX_f = widthf / 2;
    cenY_f = heightf / 2;

    int amp, cout;
    cout   = 11;
    amp    = 2048;
    //amp    = pow(2, cout);
    //printf("cout: %d , amp: %d\n", cout, amp);

    BYTE outimg[heightf * widthf];
    for(i = 0;i < heightf; i++) {
        for(j = 0;j < widthf; j++) {
            outimg[i * widthf + j] = 0;
            Xf  =    i - cenX_f;
            Yf  =    j - cenY_f;
            fXp =    cos(angle) * Xf - sin(angle) * Yf + cenX_p;    //对应原图横坐标
            fYp =    sin(angle) * Xf + cos(angle) * Yf + cenY_p;    //对应原图纵坐标
            Xp  =    (int)fXp;
            Yp  =    (int)fYp;
            //printf("%f  %f\n", fXp - Xp, fYp - Yp);
            /*
            将浮点数扩大amp倍，化成整数，减少浮点数的运算
             */
            int u = (fXp - Xp)*amp; //浮点坐标小数部分，左移7位
            int v = (fYp - Yp)*amp;
            //printf("%d %d\n", u, v);
            /*
            f(i+u,j+v) = (1-u)(1-v)f(i,j) + (1-u)vf(i,j+1) + u(1-v)f(i+1,j) +
            uvf(i+1,j+1)
            再右移7+7=14位，还原原来扩大的倍数
             */
            if(Xp >= 0 && Xp < height && Yp >= 0 && Yp < width) {//在原图范围内
              outimg[i * widthf + j] = ( (amp-u)*(amp-v)*   orimg[Xp * width +
Yp] +
              (amp-u)*v* orimg[Xp * width + Yp+(Yp == width - 1 ? 0 : 1 )] +
              u*(amp-v)* orimg[(Xp+(Xp == height - 1 ? 0 : 1 )) * width + Yp] +
              u*v*       orimg[(Xp+(Xp == height - 1 ? 0 : 1 )) * width + Yp+(Yp
==
              width - 1 ? 0 : 1 )] ) >> (cout*2);
            }
```

```
        }
        //rotate部分计算很慢，可以打印进度作为提示
        //if(i%18==0){printf("Rotating...%d%\n", i/18*10);}
    }
    //printf("Finish rotating.\n");

    /*
    Usage: output image
    Varibles:   outimg:  store the image data used for output
                q:       output image path(pointer)

     */
    if( ( fp = fopen( q, "wb" ) ) == NULL )
    {
        printf("Can't create the image at %s\n", q);
        return 0;
    }

    for( i = 0; i < heightf; i++ ) {
        fwrite( &outimg[i*widthf], 1 , widthf, fp );  //按行输出
    }
    printf("The image is written to %s\n", q);
    fclose(fp);

    finish=clock();    //结束
    time=(double)(finish-start)/CLOCKS_PER_SEC;//计算运行时间
    printf("time=%lf\n",time);//输出运行时间

}
```

参考链接

使用 ARM 处理器顺时针旋转 45 度灰度图像 | 槐雪

图像处理之双线性插值法Brandon懂你的博客-CSDN博客图像线性插值

C语言左移(<<)和右移(>>)蓝海洋高飞的博客-CSDN博客_c语言的左移函数