数值分析实验

基于Lagrange插值的图像缩放算法的实现

## 文档信息

|  |  |
| --- | --- |
| 源代码 | 可执行文件 |
| 有 | 有 |

## 学术诚信承诺书

本人承诺本课程设计提交的所有源代码和文档不存在抄袭、剽窃行为。如果被他人发现存在抄袭、剽窃、雷同等学术不端行为，愿意承担所带来的一切后果。

## 题目概述

1. 概述

本实验考查Lagrange插值的应用，采用Lagrange插值对BMP图像进行缩放处理。实验要求如下。

1. C语言编程。
2. 能够指定待缩放图像。
3. 能够指定目标图像尺寸。
4. 能够指定插值所需样本点数。
5. 能够保存缩放图像。
6. 所要完成的任务说明

|  |  |  |
| --- | --- | --- |
| 序号 | 功能分类 | 任务 |
| 1 | 必做 | 灰度图像的缩放 |
| 2 | 选作 | 彩色图像的缩放 |

1. 完成情况

|  |  |
| --- | --- |
| 功能序号 | 完成情况说明 |
| 1 | 完成 |
| 2 | 完成 |

1. 开发环境说明

本C程序使用Dev-C++软件编写，软件Version 5.11。

## 程序分析与设计

1. 系统分析

本程序创建函数函数及其功能、参数、返回值如下。

1. void Lagrange

(uint16 bpp,uint8 data[],uint8 y[],uint32 source\_Width,uint32 source\_Height,uint32target\_Width,uint32 target\_Height,uint32 num\_SamplePoint,uint32 stride\_Source)

函数功能：对图像像素数组使用Lagrange插值方法得到缩放后的像素数组。

参数含义：bpp（图像类型，bpp=8表示灰度图，bpp=24表示彩色图片）。

data[]（源图像像素数据）。

y[]（缩放后的目标图像像素数据）。

source\_Width（源图像宽度）。

source\_Height（源图像高度）。

target\_Width（目标图像宽度）。

target\_Height（目标图像高度）。

num\_SamplePoint（插值所需样本点数）。

stride\_Source（源图像每行元素的个数）。

函数返回值类型：void。

1. BMP\_HEADER get\_Image(char image\_Name[],uint8 data[])

函数功能：读取指定bmp文件的文件头、信息头、调色板（灰度图）、图像数据信息，并返回文件头和信息头。

参数含义：image\_Name[]（源图像文件名）。

data[]（源图像像素数据）。

函数返回值类型：BMP\_HEADER（自定义数据结构，用于存储bmp文件头和信息头）。

1. void write\_Image(char image\_Name[],uint8 y[],BMP\_HEADER header,int target\_Width,int target\_Height)

函数功能：将处理好的像素矩阵连同其他数据一起写到外存（保存bmp缩放图像）。

参数含义：image\_Name[]（目标图像文件名）。

y[]（目标图像像素数据）。

header（源图像文件头、信息头等信息）。

target\_Width（目标图像宽度）。

target\_Height（目标图像高度）。

函数返回值类型：void。

1. void print\_BMPHEADER(BMP\_HEADER header)

函数功能：在屏幕上输出bmp文件头和信息头数据。

参数含义：header（要输出的文件头和信息头数据）。

函数返回值类型：void。

1. int main()

函数功能：C程序主函数，程序开始入口。

参数含义：无参数。

函数返回值类型：int。

函数之间的调用关系图如图4.1所示。



图4.1 函数调用关系图

1. 函数设计

各函数函数流程图和代码注释。

1. Lagrange函数流程如图4.2.1所示，对映代码及注释如下。



图4.2.1 Lagrange函数流程图

Lagrange函数代码段

1. **void** Lagrange(uint16 bpp,uint8 data[],uint8 y[],uint32 source\_Width,uint32 source\_Height,uint32 target\_Width,uint32 target\_Height,uint32 num\_SamplePoint,uint32 stride\_Source) {
2. //bpp表示图像类型（8为灰度图，24为彩色图像）
3. //data用来存储像素信息
4. //source\_Width,source\_Height,target\_Width,target\_Height分别表示源图像宽度，源图像高度，目标图像宽度，目标图像高度
5. //num\_SamplePoint表示插值所需样本点数，stride\_Source表示像素矩阵的列数
6. **if**(bpp == 8) {
7. //处理灰度图
8. **int** i = 0,j = 0,stride\_target = (target\_Width+3)/4\*4,p = 0;
9. **for**(i=0;i<target\_Height;i++) {
10. **static** **int** set[2048\*2048\*3] = {0};
11. **static** **double** g[2048\*2048\*3] = {0.0};
12. **for**(j=0;j<stride\_target;j++) {
13. //对每个像素点进行拉格朗日插值
14. g[j] = g[0] + (**double**)j\*(**double**)(stride\_Source-1)/(**double**)(stride\_target-1);
15. **int** floor;
16. //求距离该点最近的指定个数的样本点数组下标
17. **if**(num\_SamplePoint%2==0) {
18. floor = (**int**)g[j];
19. floor -= (num\_SamplePoint-2)/2;
20. }
21. **else** {
22. floor = round(g[j]);
23. floor -= (num\_SamplePoint-1)/2;
24. }
25. **for**(p=0;p<num\_SamplePoint;p++) {
26. set[p] = floor;
27. floor++;
28. }
29. **double** temp = 0;
30. **int** flag = 0;
31. **for**(p=0;p<num\_SamplePoint;p++) {
32. **double** l = 1.0;
33. **int** q = 0;
34. //计算拉格朗日插值因子
35. **for**(q=0;q<num\_SamplePoint;q++) {
36. **double** a=set[q],b=set[p];
37. **if**(a!=b) {
38. l = l \* (g[j]-a)/(b-a);
39. }
40. }
41. **double** a=source\_Height,b=target\_Height;
42. **double** c = (**double**)i \* (**double**)(a - 1) / (**double**)(b - 1);
43. **int** index = round(c);
44. **if**(set[p] < 0) {
45. set[p] = 0 - set[p];
46. flag = 1;
47. }
48. index = index \* stride\_Source + set[p];
49. //插值求和
50. temp += l\*(**double**)data[index];
51. **if**(flag==1) {
52. set[p] = 0 - set[p];
53. flag = 0;
54. }
55. }
56. //保存插值结果
57. y[i\*stride\_target+j] = (uint8)temp;
58. }
59. }
60. }
61. **else** **if**(bpp == 24) {
62. //处理彩色图片
63. **int** i = 0,j = 0,stride\_target = (target\_Width\*3+3)/4\*4,p = 0,k = 0;
64. **for**(i=0;i<target\_Height;i++) {
65. **for**(k=0;k<3;k++) {
66. **static** **int** set[2048\*2048\*4] = {0};
67. **static** **double** g[2048\*2048\*4] = {0.0};
68. **static** uint8 fictitious\_y[2048\*2048\*4] = {0};
69. //提取出一行中相同颜色的像素点（RGB）
70. **for**(j=0;j<stride\_Source;j++) {
71. **double** c = (**double**)i \* (**double**)(source\_Height - 1) / (**double**)(target\_Height - 1);
72. **int** index = round(c);
73. fictitious\_y[j] = data[index\*stride\_Source+3\*j+k];
74. }
75. **for**(j=0;j<stride\_target/3;j++) {
76. //对每个像素点进行拉格朗日插值
77. g[j] = g[0] + (**double**)j\*(**double**)(stride\_Source/3-1)/(**double**)(stride\_target/3-1);
78. **int** floor;
79. //求距离该点最近的指定个数的样本点数组下标
80. **if**(num\_SamplePoint%2==0) {
81. floor = (**int**)g[j];
82. floor -= (num\_SamplePoint-2)/2;
83. }
84. **else** {
85. floor = round(g[j]);
86. floor -= (num\_SamplePoint-1)/2;
87. }
88. **for**(p=0;p<num\_SamplePoint;p++) {
89. set[p] = floor;
90. floor = floor + 1;
91. }
92. **double** temp = 0;
93. **int** flag = 0;
94. **for**(p=0;p<num\_SamplePoint;p++) {
95. **double** l = 1.0;
96. **int** q = 0;
97. //计算拉格朗日插值因子
98. **for**(q=0;q<num\_SamplePoint;q++) {
99. **double** a=set[q],b=set[p];
100. **if**(a!=b) {
101. l = l \* (g[j]-a)/(b-a);
102. }
103. }
104. **if**(set[p] < 0) {
105. set[p] = 0 - set[p];
106. flag = 1;
107. }
108. //插值求和
109. temp += l\*(**double**)fictitious\_y[set[p]];
110. **if**(flag==1) {
111. set[p] = 0 - set[p];
112. flag = 0;
113. }
114. }
115. //保存插值结果
116. y[i\*stride\_target+j\*3+k] = (uint8)temp;
117. }
118. }
119. }
120. }
121. **else** {
122. //返回错误信息
123. perror("Wrong type of image.");
124. }
125. }

（2）get\_Image函数流程如图4.2.2所示，对映代码及注释如下。



图4.2.2 get\_Image函数流程图

get\_Image函数代码段

1. BMP\_HEADER get\_Image(**char** image\_Name[],uint8 data[]) {
2. **int** fd = open(image\_Name, O\_RDONLY | O\_BINARY);
3. BMP\_HEADER header;
4. **if**(fd == -1) {
5. ;
6. }
7. **else** {
8. //读取图像文件头和信息头
9. read(fd, &header, **sizeof**(BMP\_HEADER));
10. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*源图像\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
11. //输出文件头、信息头数据
12. print\_BMPHEADER(header);
13. //跳过调色板部分
14. lseek(fd, header.bfOffBits, SEEK\_SET);
15. //读取图像数据部分
16. read(fd, data, header.biSizeImage);
17. close(fd);
18. //返回图像文件头和信息头数据
19. **return** header;
20. }
21. }

（3）write\_Image函数流程如图4.2.3所示，对映代码及注释如下。



图4.2.3 write\_Image函数流程图

Write\_Image函数代码段

1. **void** write\_Image(**char** image\_Name[],uint8 y[],BMP\_HEADER header,**int** target\_Width,**int** target\_Height) {
2. **char** fName[256],temp[16];
3. //确定要写的文件名
4. strcpy(fName,"");
5. strncat(fName,image\_Name,strlen(image\_Name)-4);
6. itoa(target\_Width,temp,10);
7. strcat(temp,"x");
8. strcat(fName,temp);
9. itoa(target\_Height,temp,10);
10. strcat(fName,temp);
11. strcat(fName,".bmp");
12. **int** fd = open(fName, O\_WRONLY | O\_CREAT | O\_TRUNC | O\_BINARY);
13. **if**(fd < 0 ) {
14. ;
15. }
16. //写文件头和信息头
17. uint16 bpp = header.biBitCount;
18. uint32 compression = header.biCompression;
19. memset(&header, 0, **sizeof**(BMP\_HEADER));
20. header.bfType = 'M'\*256 + 'B';
21. header.bfOffBits = **sizeof**(BMP\_HEADER) + ((bpp==8)?1024:0);
22. header.biSize = 40;
23. header.biWidth = target\_Width;
24. header.biHeight = target\_Height;
25. header.biPlanes = 1;
26. header.biBitCount = bpp;
27. header.biCompression = compression;
28. header.biSizeImage = (bpp==8)?((target\_Width+3)/4\*4\*target\_Height):((target\_Width\*3+3)/4\*4\*target\_Height);
29. header.bfSize = header.bfOffBits + header.biSizeImage;
30. header.biClrUsed = (bpp==8)?256:0;
31. write(fd, &header, **sizeof**(BMP\_HEADER));
32. //写文件调色板数据
33. **if**(bpp == 8) {
34. uint8 clrPal[256\*4];
35. **int** i = 0;
36. **for**(i=0; i<256; i++) {
37. clrPal[4\*i] = clrPal[4\*i+1] = clrPal[4\*i+2] = i;
38. clrPal[4\*i+3] = 0;
39. }
40. write(fd, clrPal, 256\*4);
41. }
42. //写图像数据部分
43. write(fd, y, header.biSizeImage);
44. close(fd);
45. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*目标图像\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
46. //输出写的文件头和信息头数据
47. print\_BMPHEADER(header);
48. //结束
49. **return**;
50. }

（4）print\_BMPHEADER函数流程如图4.2.4所示，对映代码及注释如下。



图4.2.4 print\_BMPHEADER函数流程图

print\_BMPHEADER函数代码段

1. **void** print\_BMPHEADER(BMP\_HEADER header) {
2. //输出BMP\_HEADER信息
3. printf(" bfType          : %c%c\n", header.bfType%256,header.bfType/256 );
4. printf(" bfSize          : %d  \n", header.bfSize          );
5. printf(" bfReserved1     : %d  \n", header.bfReserved1     );
6. printf(" bfReserved2     : %d  \n", header.bfReserved2     );
7. printf(" bfOffBits       : %d\n\n", header.bfOffBits       );
8. printf(" biSize          : %d  \n", header.biSize          );
9. printf(" biWidth         : %d  \n", header.biWidth         );
10. printf(" biHeight        : %d  \n", header.biHeight        );
11. printf(" biPlanes        : %d  \n", header.biPlanes        );
12. printf(" biBitCount      : %d  \n", header.biBitCount      );
13. printf(" biCompression   : %d  \n", header.biCompression   );
14. printf(" biSizeImage     : %d  \n", header.biSizeImage     );
15. printf(" biXPelsPerMeter : %d  \n", header.biXPelsPerMeter );
16. printf(" biYPelsPerMeter : %d  \n", header.biYPelsPerMeter );
17. printf(" biClrUsed       : %d  \n", header.biClrUsed       );
18. printf(" biClrImportant  : %d  \n", header.biClrImportant  );
19. printf("\n");
20. //输出完毕
21. **return**;
22. }

（5）main函数流程如图4.2.5所示，对映代码及注释如下。



图4.2.5 main函数流程图

main函数代码段

1. **int** main() {
2. BMP\_HEADER header;
3. **int** fd = -1;
4. **char** image\_Name[256];
5. **static** uint8 data[2048\*2048\*4],y[2048\*2048\*4];
6. uint32 target\_Width = 0,target\_Height = 0,num\_SamplePoint = 0,stride\_Source = 0;
7. **while**(1) {
8. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
9. printf("请输入待缩放的图片名(输入0结束程序)：");
10. //输入要缩放的图像名
11. scanf("%s",image\_Name);
12. **if**(strcmp(image\_Name,"0") == 0) {
13. //结束
14. **return** 0;
15. }
16. printf("请指定目标图像尺寸(宽度、高度)：");
17. //输入目标图像尺寸
18. scanf("%d%d",&target\_Width,&target\_Height);
19. printf("请指定插值所需样本点数：");
20. //输入插值所需样本点数
21. scanf("%d",&num\_SamplePoint);
22. //读取待缩放图片
23. header = get\_Image(image\_Name,data);
24. //计算待缩放图像每行的元素个数
25. stride\_Source = header.biSizeImage / header.biHeight;
26. //进行拉格朗日插值
27. Lagrange(header.biBitCount,data,y,header.biWidth,header.biHeight,target\_Width,target\_Height,num\_SamplePoint,stride\_Source);
28. //将处理好的图像数据保存
29. write\_Image(image\_Name,y,header,target\_Width,target\_Height);
30. }
31. }

## 测试分析

1. 测试与调试

代码测试环节主要测试灰度图、彩色图的图像放大缩小功能是否正常(放大缩小后的尺寸为4的整数倍与不是4的整数倍两种情况)。这里测试了八种情况，分别为灰度图缩小为128x128,145x179尺寸，放大为512x512，511x411尺寸，彩色图缩小为200x200，125x119尺寸，放大为256x512，711x699尺寸。

1. 测试结果分析

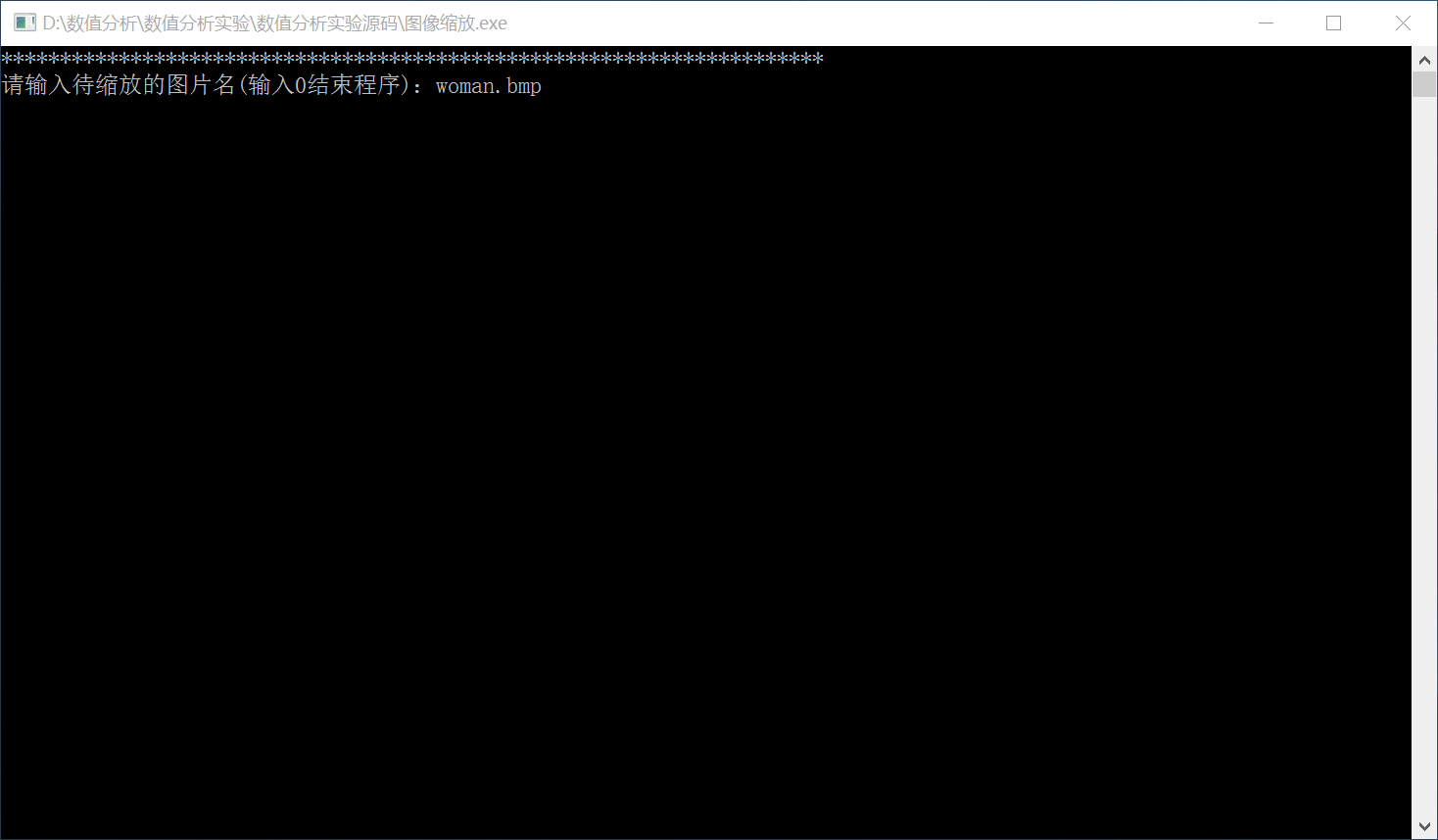
经检验，以上八种测试情况输出结果均正常，缩放后的图片已保存在提交文件中。

1. 结论

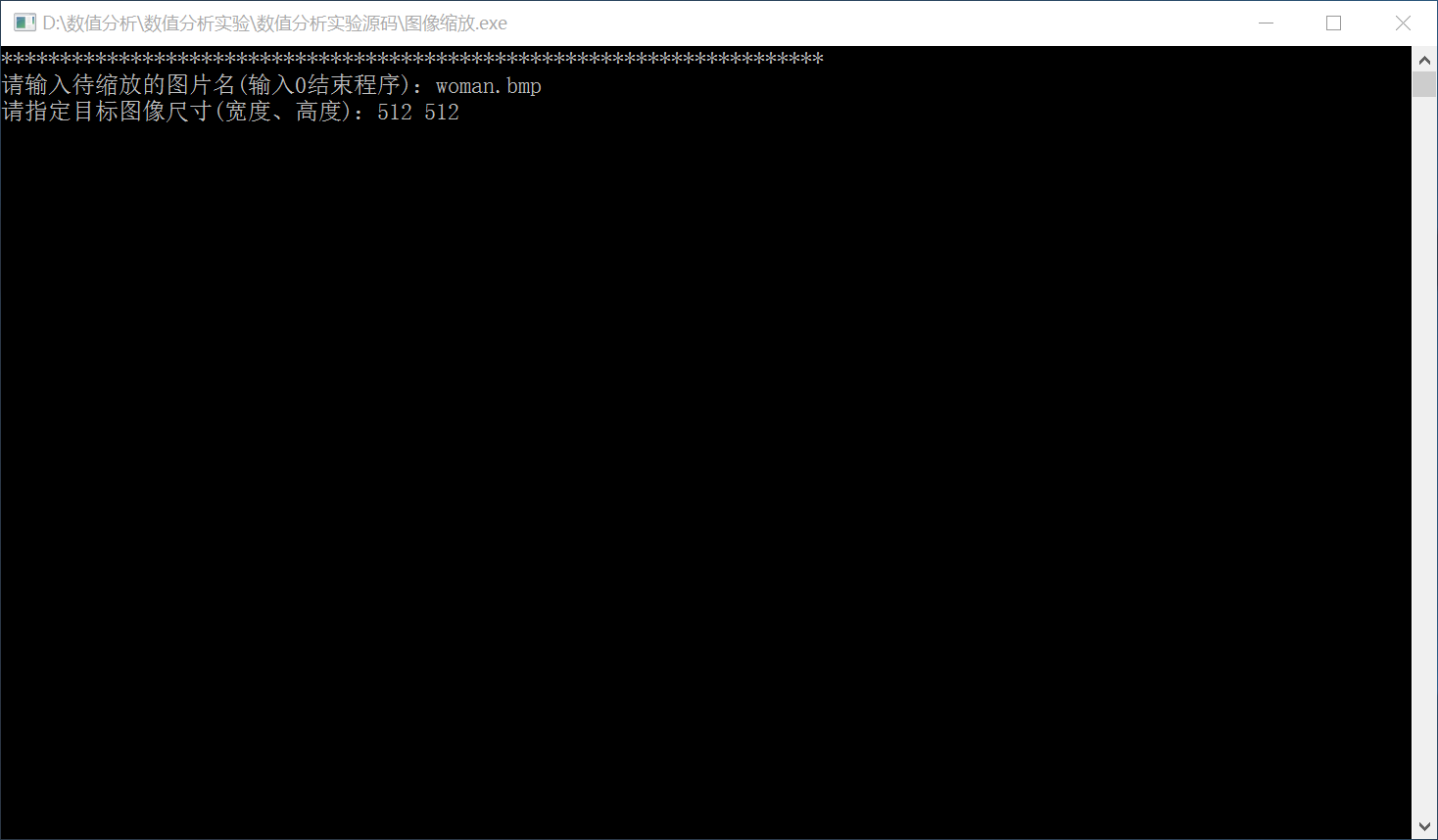
源码程序能够完成对灰度图和彩色图的缩放功能，且能满足目标图像尺寸不是4的整数倍情况下的缩放要求。

## 系统演示

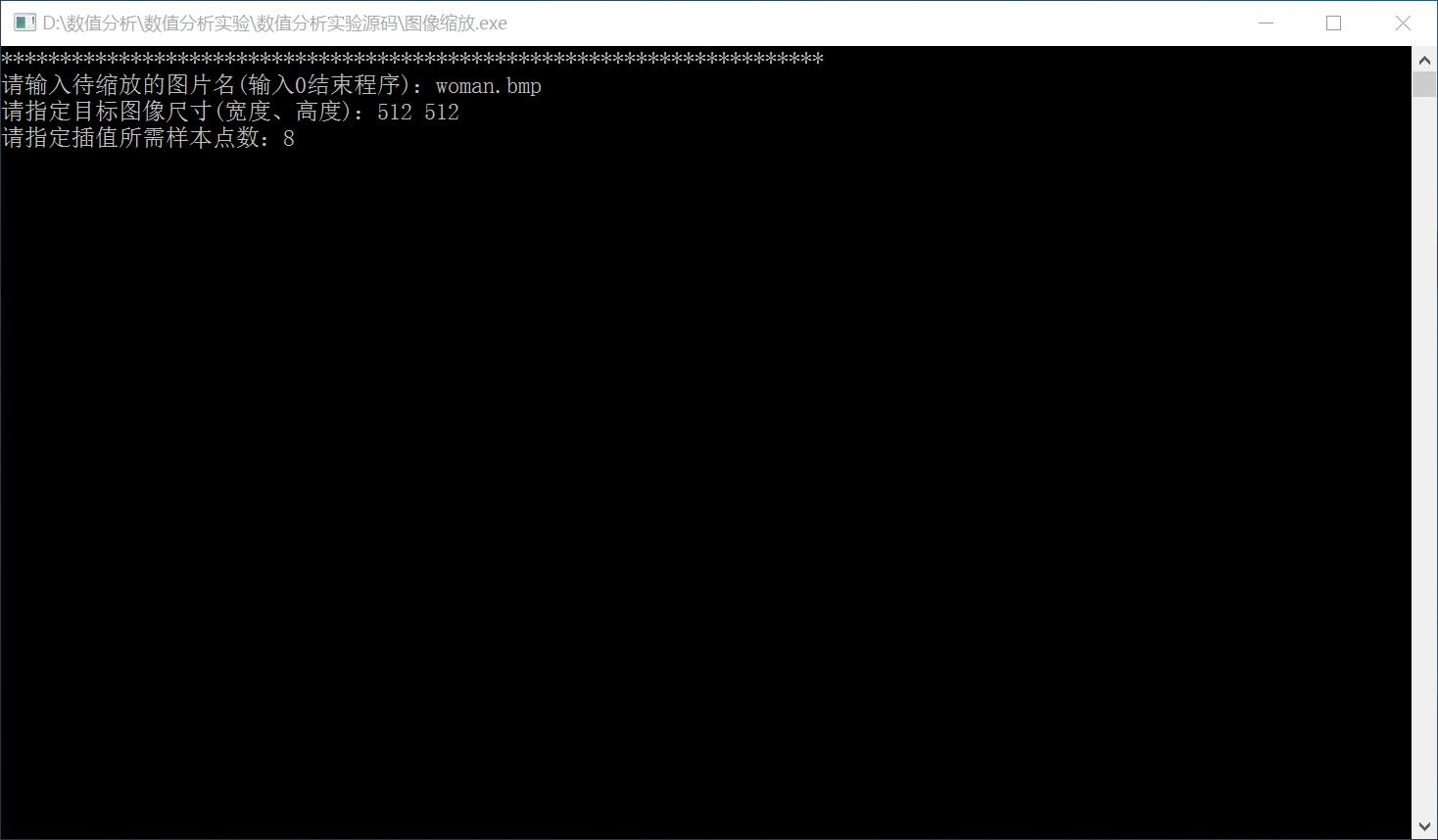
1. 系统个功能运行截图与说明
2. 输入待缩放的图片名（若要结束程序，则输入0）。



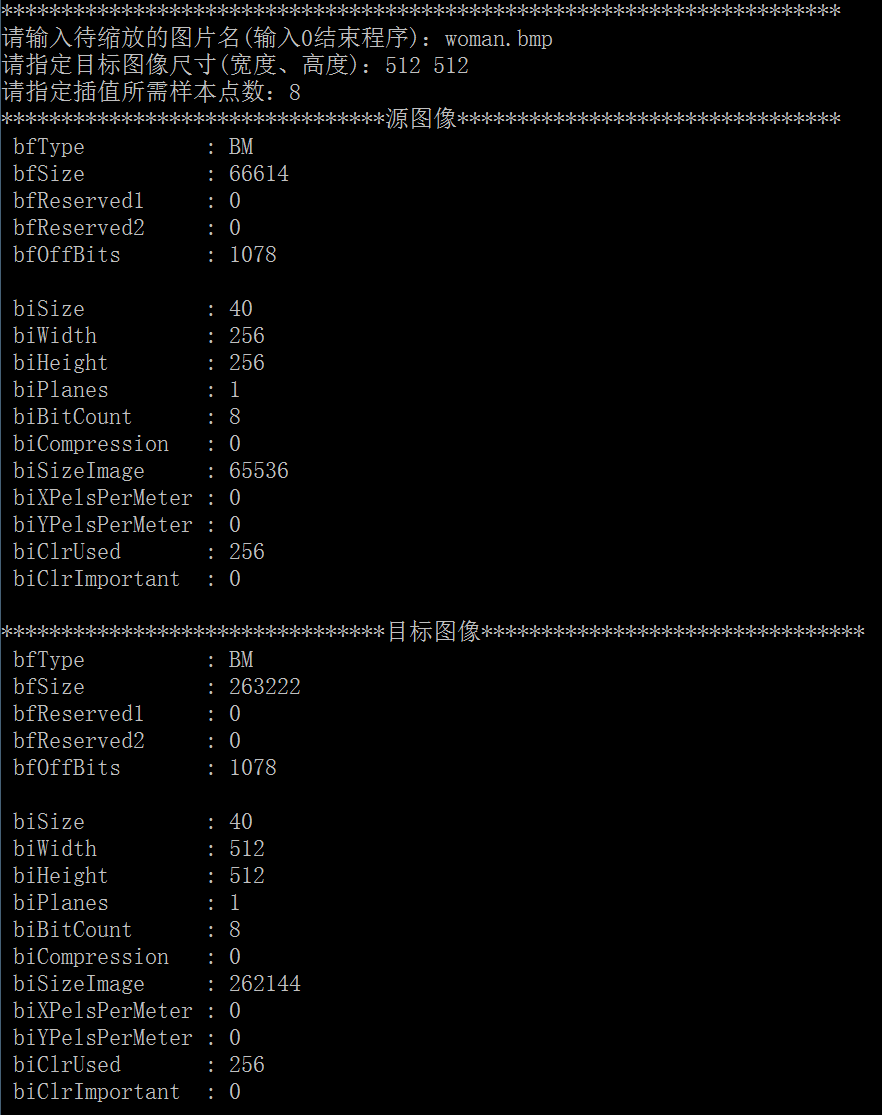
1. 指定目标图像尺寸



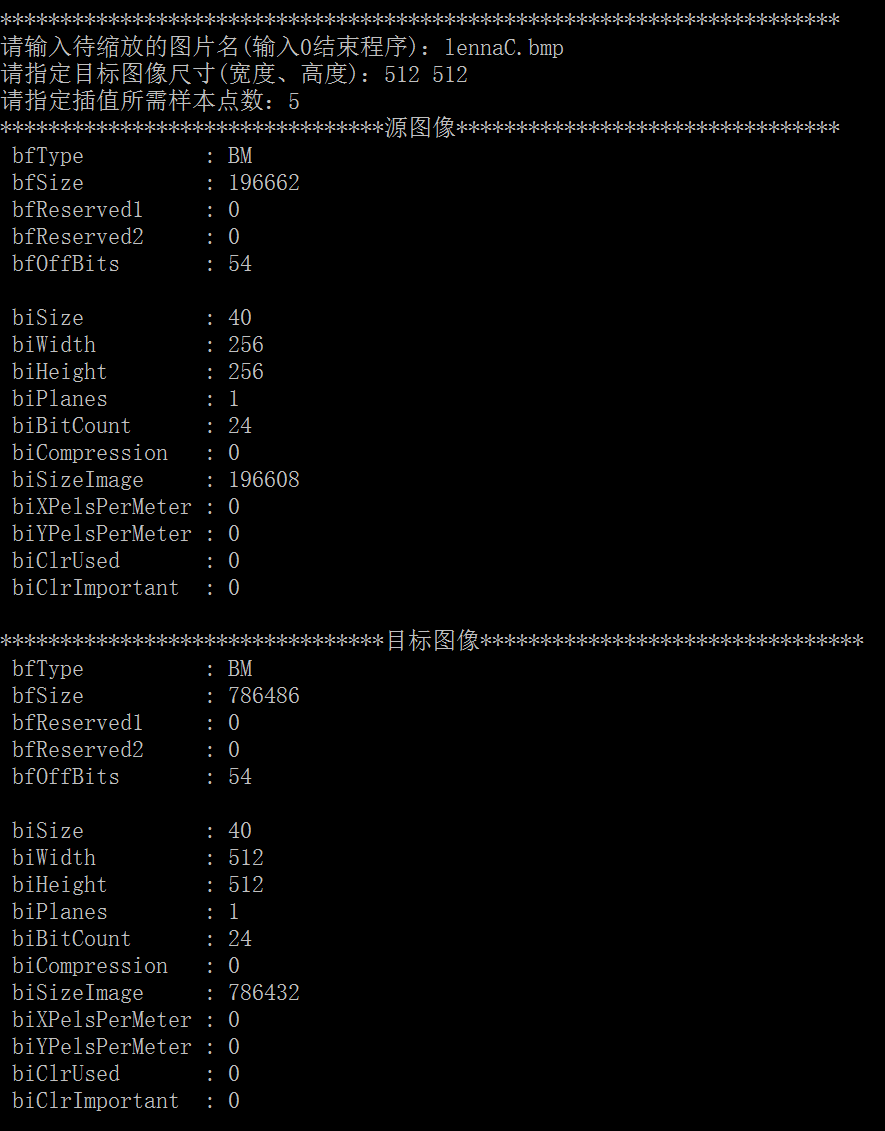
1. 指定插值所需样本点数



1. 指定样本点数完成后，会输出源图像与目的图像信息，此时缩放已经完成，可在程序所在文件夹中查看缩放后的图像。



1. 彩色图操作与灰度图类似，只需将输入待缩放的图片名换成lennaC.bmp即可。



## 设计体会

1. 设计过程中的问题及其解决方案
2. 问题：当目标图像宽度不为4的整数倍时，图像缩放后异常。

解决方案：对bmp文件存储每行图像数据元素数量进行/4\*4，保证每行像素点个数为4的整数倍。

1. 问题：图像边界处异常，且指定的插值所需样本点数越多，边界异常区域越大。

解决方案：只将样本值f(x)进行对称延拓，而不是将x和f(x)都进行对称延拓。

1. 设计过程中的错误及其原因分析
2. 问题：当目标图像宽度不为4的整数倍时，图像缩放后异常。

原因分析：由于bmp图像像素数据实际以一维数组存储，而显示图片时每行像素点个数为4的整数倍。所以如果存储时每行像素点个数不为4的整数倍，显示出来会产生扭曲。

1. 问题：图像边界处异常，且指定的插值所需样本点数越多，边界异常区域越大。

原因分析：最初设计时将x和f(x)都进行了对称延拓，导致了插值结果过大和过小情况的出现，正确做法应是只将样本值f(x)进行对称延拓，x不变。

1. 体会和收获

此次实验代码编写花费了我大概两天的时间。通过此次实验，我对拉格朗日插值以及图像处理有了更加深刻的理解，同时也在一定程度上提高了自己构建代码结构，debug的能力，令我受益很多。