

CS219 Project4: Image Processing Library

蔡易霖 12312927

April 2025

目录

1	不同类型的图像输入	2
1.1	基于现有库的图像输入	2
1.2	size_t 类型输入的设计	3
2	类的设计	3
2.1	类似智能指针的实现	4
2.2	内存管理和内存拷贝的对比	4
2.3	内存问题	4
2.4	错误处理	5
2.5	分化的类	6
3	功能实现	6
3.1	增加亮度	6
3.2	融合图像	7
4	综合功能测试	8
4.1	错误的图片输入和图像构建	8
4.2	最终测试	10

开始的思考

拿到这次的 project 的题目后，我惊讶地发现这次的题目和上次的十分相似。但是如此相似的情况下，对于这次的 project，我又应该多做些什么呢？为此，我仔细比较了两次 project 的题目，发现了这次 project 和上次 project 还是有一些不同的：

最显眼的不同是语言从 c 变成了 c++，并且要求我们写一个类。这一点的具体实现倒是很明确的。

同时我意识到上次的要求是操作 bmp 文件，然而这一次没有指定输入图像的类型，或许我应该识别输入图像的类型并且针对不同的类型使用不同的输入方式？

同时这一次不需要很多图像处理操作了，而是应该针对图片处理操作的质量。

明确了这几点后，我想我可以从不同类型的图像的输入入手了。

1 不同类型的图像输入

1.1 基于现有库的图像输入

开始时，我本想自己手动实现不同类型的图像输入，但后来我发现，对于像 jpg 这样格式的文件，其加密格式非常的复杂。同时我发现有现成的 stb_image 库可以达到不同类型的图片的输入的作用。既然如此，为什么我不直接使用呢？

但是在了解了之后，我发现需要把 stb_image.h 和 stb_image_write.h 文件放到文件夹中才能调用其中的输入函数。考虑到提交报告的时候只能提交一个.h 文件，我认为我有必要说明：**要调用此图像编辑库，需要从 <https://github.com/nothings/stb> 仓库找到并新下载 stb_image.h 和 stb_image_write.h**

在了解到了这件有力的工具之后，我只需要调用已有的函数就能实现不同类型的图片的输入的功能了。虽然它实现起来似乎有一些慢，但是 IO 操作应当是频率很低的操作，并且人家已然封装好，因此我接受并且使用它作为我的输入工具。

1.2 size_t 类型输入的设计

然而这样就迎来了一个问题：现有的 `std_image` 库的数据输出是 `int` 类型的，但是我希望能够使用 `size_t` 类型的变量来决定图片的长宽。因此，我在 `Image` 类的输入和输出函数中都添加了类型转换的模块，并且在输入或要输出的图片过大的时候进行报错。在这样的基础上，我的程序能够支持长宽大小为 2147483647 以内的图片。其图片大小为 10^{20} 量级的，也就是大约 $10^{11}Gb$ 。我认为，作为一张图片，支持这样的图片大小已经相当足够。

2 类的设计

拿到题目之后，我先把上次的 `project` 的代码丢给 DeepSeek，让它模仿功能的实现帮我生成了一个框架。运行了它给我 `idlibrary` 之后，我发现增加亮度和融合图片的功能都能正确地完成了。

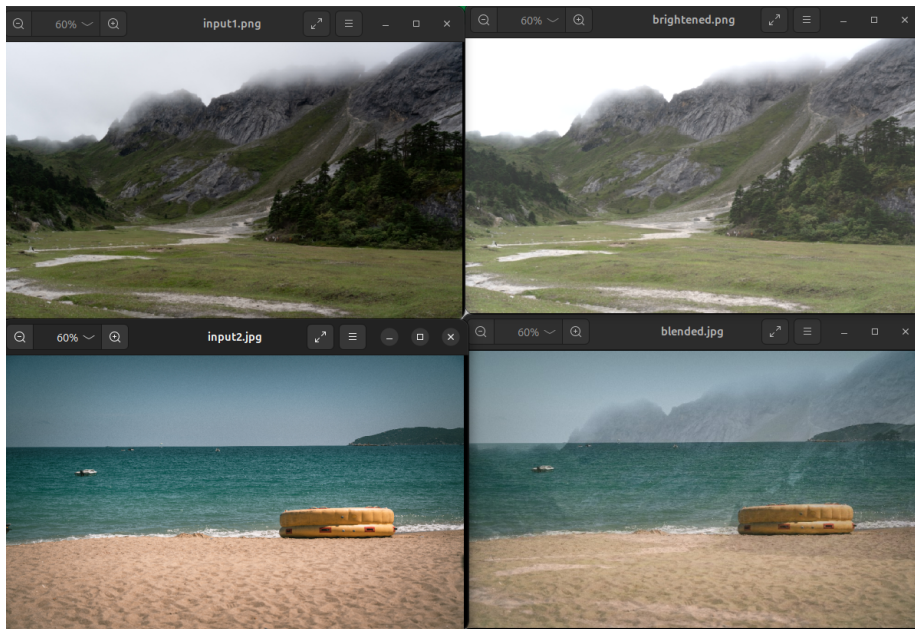


图 1：如图，左侧是输入的图片，右侧是操作得到的图片

然而，在我仔细观察了代码之后，我发现 DeepSeek 是通过直接复制数据的方式避免 `double free` 的，也就是硬拷贝，这对于数据量很大的图片处

理来说效率是十分低下的。因此，我想要在保证内存安全的方式上进行修改。

2.1 类似智能指针的实现

由于我并不喜欢智能指针，我选择了使用一个指针指向被引用量的方式保证内存的安全，也就是软拷贝。每次多出一个指向一个图片的赋值，我就将这个图片的被引用量加一。每次图片被释放的时候，我就将被引用量减一，只有在被引用量归零的时候我才会清除这块内存。

2.2 内存管理和内存拷贝的对比

在经过了修改之后，我实现了一样的效果。由于这一部分的优化主要体现在复制图片操作运行的时间上，我分别运行了优化前后的代码，分别比较了它们的速度。

图像大小	硬拷贝时间/ms	软拷贝时间/ms
1000 × 600	0.596166	0.000212
2000 × 1200	2.193394	0.000207
5000 × 3000	13.741700	0.000205
10000 × 6000	51.280043	0.000202
15000 × 9000	116.451042	0.000198
20000 × 12000	209.701027	0.000204

表 1: 运行时间

可以看到，经过修改之后，复制图片的操作消耗的时间远远降低了。观察到硬拷贝消耗的时间随着图片大小以 n^2 的大小增大，这和其完整复制图片每个像素的操作相符合；而软拷贝时间不随时间大小变化而变化，并且非常小，这是因为其只操作了指针，而没有操作内存本身。因此，这番优化大大提升了程序的运行效率。

2.3 内存问题

在修改完代码之后，我使用工具 `fsanitize=address` 检测内存泄漏，却惊讶地发现我看似严丝合缝的代码居然还有内存问题！然而，这次的报错却

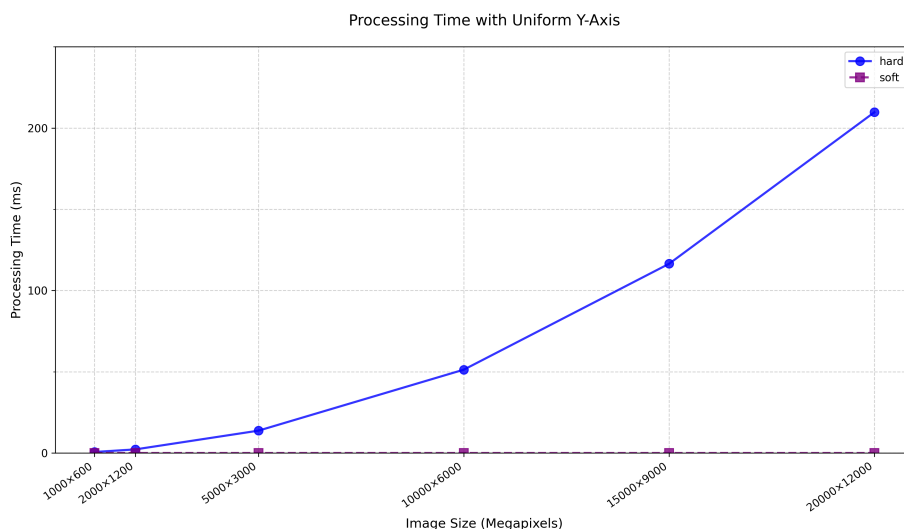


图 2: 时间消耗对比

和以往不同，不是一个内存泄漏的问题：

```
SUMMARY: AddressSanitizer: alloc-dealloc-mismatch ../../../../src/libsanitizer/a
san/asan_new_delete.cpp:155 in operator delete[](void*)
==64297==HINT: if you don't care about these errors you may set ASAN_OPTIONS=all
oc_dealloc_mismatch=0
==64297==ABORTING
```

图 3: 报错信息如下

在询问了 DeepSeek 之后，我发现问题在于我的内存分配方式和释放方式不匹配。在我原来的代码里面，DeepSeek 写的部分用了 new 分配内存，而我使用了 malloc。而在 c++ 中，malloc 分配的内存要用 free 释放，而 new 分配的内存要用 delete 释放。在修改了这一点之后，我的代码果然不会有内存问题了。

2.4 错误处理

我不得不考虑到用户可能不止会输入正确的图片，还可能创造一些长宽为负，类型古怪，或者导入一些不存在的图片等错误操作。面对这些操作的时候，我应保证我的 library 不会崩溃。于是针对宽，高，通道数的构造方法，和导入本地数据的构造方法，我对不合法的输入直接进行了 throw 的

报错。这样会直接中断程序的进行，是很合理的。毕竟如果数据错误，就不应该继续让用户继续操作了。

至此，我们就完成了兼容不同类型，不同通道数的图片类的基本设计。

2.5 分化的类

然而，我认为还有一些可以做的事情。我意识到如果我直接使用参数创建一张图片，这张图片的类型会是未知的，这一点将会不利于程序的进一步拓展。为此，我创建了 `JpegImage`，`PngImage` 和 `GrayscaleImage` 三个子类，对程序创造的图片进行进一步分辨。

同时，我意识到对于一些比较小的图片，其实并不需要 `size_t` 的空间进行数据的装载。因此，我创造了 `Image256` 类，其他操作与 `Image` 类相同，但是可以只用 `unsigned char` 类型装数据，可以大大节省空间。

在进行这样的分化之后，我相信我的程序会变得更加用户友好。

3 功能实现

3.1 增加亮度

增加亮度的操作其实很简单，只需要对于图片的每一个通道的每一个参数增加对应的数值就好了。比较关键的点应当是效率的优化与操作的鲁棒性。于是，我使用了 `SIMD` 和 `openMP` 来进行程序运行速度的优化。

进行了优化之后，我依然是将运行前后的程序分别运行并比较运行速度。在上次的 `project` 中，为了测试不同实现方式的运行速度，我使用了一个外置的 `python` 程序，调用我的 `cpp` 程序并进行计时。然而，在这次的 `project` 中，由于我写的是 `library`，我可以直接写一个 `demo.cpp`，调用 `library` 并进行计时。我还是使用了 `clock_gettime` 来获得精确的时间。以下是我的程序优化前后的时间消耗：

可以发现这样优化前后的时间都与图片的大小正相关，并且图片尺寸越大优化效果越明显。这是因为 `SIMD` 和 `openMP` 并行的优势随着运算的数据量越来越大而越来越明显，十分合理！

图像大小	优化前时间/ms	优化后时间/ms
1000×600	0.07	0.06
2000×1200	2.05	0.22
5000×3000	11.45	4.07
10000×6000	42.46	15.65
15000×9000	92.14	32.2
20000×12000	158.94	56.72

表 2: 运行时间

3.2 融合图像

我的图像的融合是对图片的每一个通道的每一个像素进行加和求平均，因此其会对每一个长宽或者通道数不相同的图片组合报错。实现这个功能的时候，我与上一个 project 不同的地方在与添加了一个 0 到 1 之间参数，代表原图片在新图片中的占比。同样，我也对我的融合图像功能测试了运行时间。

图像大小	优化前时间/ms	优化后时间/ms
1000×600	0.92	0.21
2000×1200	4.05	1.01
5000×3000	21.51	4.66
10000×6000	83.07	17.27
15000×9000	143.55	38.01
20000×12000	329.75	66.40

表 3: 运行时间

可以发现，优化前后的程序的运行时间都随着图片的边长以接近 n^2 的关系正相关增加，也就是和图片的大小正相关。注意到优化前在图片较大的时候运行时间增加明显，而优化后的运行时间增加却不明显。这可能是因为随着数据量增大，SIMD 和 openMP 的优化效果增强，也就是程序“越跑越快”了，这一点令人比较满意。

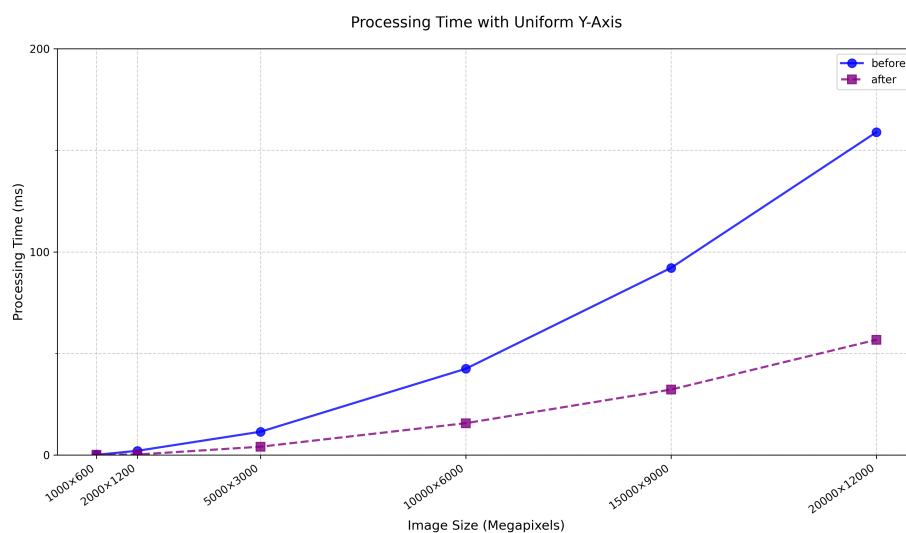


图 4: 优化前后增加亮度操作时间图像

4 综合功能测试

在完成了单独功能的实现之后，我设计了一些错误的类型来检验程序对于错误输入的反应。

4.1 错误的图片输入和图像构建

我准备了长宽为负数，通道数为负数，以及不存在的图像作为错误输入。不出所料，我的程序正常地报错了。

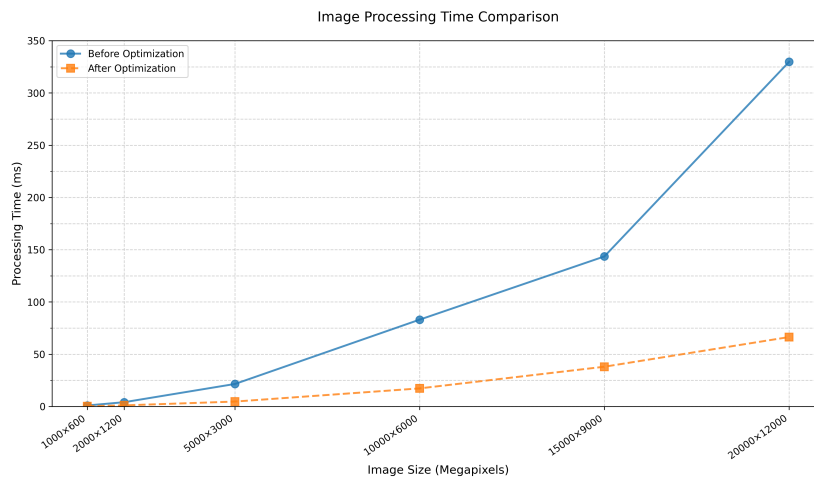


图 5: 图像融合优化前后消耗时间对比

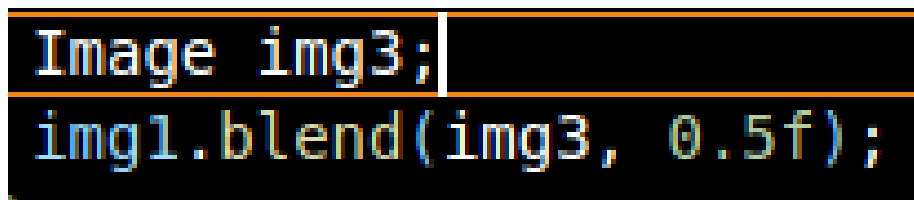
```
Image img1(-1, 2, 3); //Wrong size
Image img2(1, 2, 10); //Wrong tunnels
Image img3;
img3.load("NotExist.jpg"); //Not exist image
```

图 6: 错误的代码

```
e0isa5@e0isa5-NbF-XX:~/c++/ysq/proj4$ ./demo
Error: Invalid image dimensions when creating Image
```

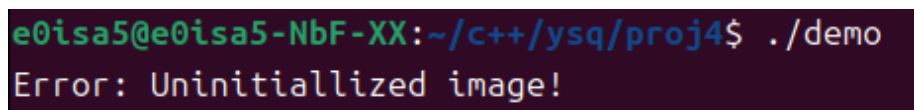
图 7: 程序正常报错

同时，我还要避免用户莫名其妙将错误的图片保存进了 Image 类，然后进行操作导致操作错误，或者将还没有初始化的图片进行操作。我对此也进行了程序处理，保证其不会莫名其妙崩掉。以下是对该处理的验证：



```
Image img3;  
img1.blend(img3, 0.5f);
```

图 8：错误的代码



```
e0isa5@e0isa5-NbF-XX:~/c++/ysq/proj4$ ./demo  
Error: Uninitiallized image!
```

图 9：程序正常报错

由此，我认为我的程序可以正确地处理错误的问题。

4.2 最终测试

想要测试我的程序的错误是一件令我十分苦恼的事情，比较要想出 bug 才能矫正 bug。但是我灵光一现：为什么不让 DeepSeek 帮我写测试程序呢？

因此，为了测试这个 library 对于引用它的用户的使用，我让 DeepSeek 帮我写了一串测试代码，并且尽可能测试到每一个函数。最终，其运行结果均为正常。

值得一提的是，他还给了我一些启发：他用了一个时间戳生成器，为生成的图片命名为时间戳的名字。这让我意识到，我可以为我的 save 函数添加一个默认函数，在没有输入目标文件名的时候将生成的文件命名为时间戳。实在是感谢 DeepSeek！



图 10: Deepseek 帮我写测试程序

```
=== Constructor/Copy Test ===
Empty image dimensions: 0x0
Created blank image: 800x600 channels: 3

=== File I/O Test ===
Loaded images:
  input1: 10000x6000
  input2: 10000x6000

=== Copy Semantics Test ===

=== Brightness Adjustment Test ===
Brightness adjustment 50 completed, time: 8ms
Brightness adjustment -100 completed, time: 8ms
Brightness adjustment 300 completed, time: 8ms

=== Image Blending Test ===
Blending alpha 0.2 completed, time: 17ms
Blending alpha 0.5 completed, time: 13ms
Blending alpha 0.8 completed, time: 13ms

=== Exception Test ===
Caught exception: Invalid image dimensions when creating Image
Caught blending exception: Images must have the same dimensions

=== Performance Test ===
Brightness adjustment for 2000x2000 image took: 4ms
Brightness adjustment for 2000x2000 image took: 0ms

All tests passed!
```

图 11: 测试结果

由于我使用了 `-fsanitize=address` 并且没有报错，我相信，我的程序已经基本没有什么问题了！

结语

这次 project 其实对于我的思考量并没有特别大？主要是有了上一次 project 的完成给了我一个模板可以对照着写，并且写 c++ 的类的问题其实还挺明确的，在 DeepSeek 的帮助下并不容易卡死解决不了。

不过，这次 project 给我的满足感还是很强的。在我印象里，这是我第一次使用 `std` 之外的 namespace。虽然理论上早就明白自己写的其他的 library 就应该这样用，但是实际用了一下的感觉还是完全不一样的。

我会将工具函数放到 github 上 <https://github.com/10a5/CS219-Advanced-Programming>