厦门大学嘉庚学院

本科生毕业论文（设计）

题　　目：基于 C++和 QT 的数字图像处理系统

姓　　名：郑添

院　　系：信息科学与技术学院

专　　业：软件工程

年　　级：2020 级

学　　号：MEE20039

指导教师：黄彪　　职称：高级实验师

2024 年 4 月 22 日

原创性声明

兹呈交的学位论文（设计），是本人在导师指导下独立完成的研究成果。除文中已经明确标明引用或参考的内容外，本论文（设计）不包含任何其他个人或集体已经发表或撰写过的研究成果。本人依法享有和承担由此论文而产生的权利和责任。

声明人（签名）：

日期： 2024 年 4 月 22 日

基于 C++和 QT 的数字图像处理系统

【摘要】 数字图像处理是计算机视觉领域的重要分支，其应用涵盖了医学影像、图像识别、图像增强等多个领域。因此，本系统设计实现了一个《基于 C++和 QT 的数字图像处理系统》。本系统以 C++为后端开发语言，采用原始代码实现，不依赖于任何第三方图像处理库。借助 QT 作为前端界面框架，实现了一个完整的数字图像处理系统。系统的重点研究内容是通过原始的 C++代码实现各种图像处理功能算法，同时利用 QT 提供的界面设计工具，使用户能够方便地使用这些功能。通过原始的 C++代码实现图像处理功能可以保证系统的稳定性和灵活性，同时利用 QT 提供的界面设计工具可以提高系统的易用性和可操作性。因此，本人所设计的基于 C++和 QT 的数字图像处理系统是一个全面而实用的工具，能够满足用户在数字图像处理方面的各种需求。

【关键词】 C++ QT 数字图像处理

Digital Image Processing System Based on C++ and QT

[Abstract] Digital image processing is an important branch in the field of computer vision, and its applications cover many fields such as medical imaging, image recognition, and image enhancement. Therefore, this system designs and implements a "Digital Image Processing System Based on C++ and QT". This system uses C++ as the back-end development language, is implemented using original code, and does not rely on any third-party image processing library. With QT as the front-end interface framework, a complete digital image processing system is implemented. The key research content of the system is to implement various image processing function algorithms through original C++ code, and at the same time use the interface design tools provided by QT to enable users to use these functions conveniently. Implementing image processing functions through original C++ code can ensure the stability and flexibility of the system, while using the interface design tools provided by QT can improve the ease of use and operability of the system. Therefore, the digital image processing system based on C++ and QT designed by me is a comprehensive and practical tool that can meet the various needs of users in digital image processing.

[Keywords] C++, QT, Digital image processing

目　　录

引言

数字图像处理技术在当今科技领域中扮演着日益重要的角色，成为了多个领域中不可或缺的一部分。随着数字图像处理应用的不断拓展和深入，研发一款高效、灵活且功能全面的数字图像处理系统显得尤为迫切。

本文围绕着“基于 C++和 QT 的数字图像处理系统”展开研究，致力于通过原始的 C++代码实现各种图像处理功能，并借助 QT 前端框架提供用户友好的界面设计。在这一系统中，本人将通过深入探讨图像读取与保存、灰度转换、自动对比度调整、高斯模糊、中值模糊等多项功能，为用户提供一套全面而实用的数字图像处理工具。

通过这一研究，本人旨在推动数字图像处理技术的发展，为学习数字图像处理的人员提供更便捷、高效的图像处理工具，以满足不断增长的应用需求。

第 1 章　绪论

1.1 课题背景

数字图像处理是计算机科学与工程领域中的一个重要研究方向，涉及图像获取、图像分析与理解、图像增强等多个方面。随着数字图像处理技术的不断发展，它在医学、军事、工业、娱乐等领域都得到了广泛应用。

C++作为一种高性能、面向对象的编程语言，具有强大的编程能力和灵活性，适用于开发需要高效处理的应用程序。

QT 是一套跨平台的 C++图形用户界面开发框架，提供了丰富的图形库和工具，方便开发者构建直观、美观的用户界面。

在数字图像处理领域，很多应用借助第三方图像处理库来简化开发过程，但这样的依赖可能导致系统庞大、性能不佳或者受到特定授权的限制。

因此，设计并实现一款基于 C++和 QT 的数字图像处理系统，完全采用原始代码实现，摒弃对第三方图像处理库的依赖，将为图像处理领域的软件开发者提供更多的自由度和控制权。

本课题旨在探索如何充分发挥 C++和 QT 的优势，设计并实现一款高效、灵活、易用的数字图像处理系统，以满足不同领域的图像处理需求。

通过此系统，开发者可以更深入地理解数字图像处理的基本原理，同时在应用开发中获得更大的自主权和定制能力。通过自主开发所有图像处理算法，不仅可以优化性能，还能够满足特定应用场景下的定制需求，提高系统的可移植性和可扩展性。

1.2 研究目的和意义

随着计算机技术的不断发展与广泛应用，第三方库的普及使得开发人员能够更快速地实现功能，但这也导致真正了解底层原理的开发人员相对较少。因此，通过在数字图像处理项目中不依赖第三方库的实现，能够大幅提升编程技能，深入了解底层原理，以及为之后的性能优化奠定基础。

操作图像实质上就是对内存的操作，这使本人更深入地理解计算机内存的工作原理和使用方式，为编程和优化提供了宝贵的见解和经验。这种底层思维和实践不仅加强了对编程的理解，还有助于解决更广泛的计算机科学问题。

本研究的首要目的是通过设计和实现基于 C++和 QT 的数字图像处理系统，深入探索数字图像处理的基本原理和算法。通过自主处理图像处理算法，本人能够在系统级别上理解各种图像处理技术的工作原理，为未来的研究提供深刻的基础。

在技术层面，通过摒弃对第三方图像处理库的依赖，本人旨在提高对

C++和 QT 的使用熟练度。这将促使本人更全面地理解这两种技术的内部机制，为本人在实际项目中的应用提供更大的自由度和灵活性。

此外，通过独立开发所有图像处理算法，减少对第三方库的依赖，不仅有助于提高系统性能，还能够适应特定应用场景的定制需求，从而提高系统的可移植性和可扩展性。

1.3 系统设计思想

本数字图像处理系统的设计思想围绕着以下几个核心原则展开：

1.独立实现核心算法:系统将自主实现数字图像处理的核心算法，包括但不限于灰度处理、边缘检测、插值等。这有助于深化本人对图像处理原理的理解，并提高系统的独立性和可控性。

2.模块化架构:采用模块化设计,将整个系统划分为独立的功能模块，每个模块负责一个特定的图像处理任务。这有利于系统的可维护性、可扩展性和代码重用性。

3.用户友好的界面设计:利用 QT 框架构建直观、用户友好的图形界面（GUI）。通过界面，用户可以轻松地上传、处理和保存图像，以及调整不同算法的参数。系统的设计力求使用户在图像处理过程中获得良好的交互体验。

4.性能优化:在算法实现上注重性能优化，使用 C++语言的高效性能，以确保系统在图像处理任务中能够快速而稳定地运行。采用合适的数据结构和算法，以提高处理大规模图像的效率。

5.可移植性和跨平台性:为了增强系统的可移植性，避免对特定平台的依赖，系统可被设计成在不同操作系统上运行。这有助于更广泛地应用系统，并使其适用于多种不同的开发环境。

6.文档意义:系统将附带详细的文档,包括算法原理,这样的设计有助于促进学习，使系统成为学习数字图像处理的理想工具。

通过遵循这些设计原则，该系统旨在提供一个功能强大、易用、灵活且性能良好的数字图像处理平台，以满足用户对图像处理的各种需求。

1.4 系统研究内容

1.图像获取与加载:设计图像获取模块，支持从本地文件系统加载图像数据。此模块负责确保系统能够有效地获取图像。

2.图像处理:开发各种图像算法，如均值滤波、中值滤波、高斯滤波、边缘检测(Sobel)、最临近插值、双立方插值等，以及图像增强算法，如亮度、对比度、饱和度调整等。这些算法可以通过用户界面调整参数，以适应不同的图像处理需求。

3.用户界面设计:使用 QT 框架设计直观的图形用户界面,包括图像展示区域、操作按钮和参数调整控件。用户可以通过界面上传图像、选择算法、调整参数并查看处理结果。

4.性能优化:对系统进行性能优化，包括多线程处理方面，以确保能够保持高效的性能。

5.跨平台适配:确保系统能够在不同操作系统上运行，通过对平台相关的问题进行处理，提高系统的可移植性。

6.文档撰写:编写详细的技术文档，包括算法原理，以方便其他开发者理解系统的设计和使用方法。这有助于推动系统的研究.

通过对这些研究内容的全面实现,该系统旨在为用户提供一个全面而自主的数字图像处理平台，涵盖了图像处理的各个方面，从而满足不同用户在图像处理领域的需求。

第 2 章　开发技术及运行环境介绍

2.1 前端 QT 技术

选择使用 QT 作为前端界面的主要原因有几个关键方面:

首先，QT 具有卓越的跨平台性，能够在不同操作系统上实现一致的用户体验，包括 Windows、Linux 和 macOS，为系统的灵活性和可移植性提供了强有力的支持。

其次，QT 提供了强大的图形库和控件，使得用户界面的设计和实现相对简单，能够创建直观、美观且功能完善的界面。由于 QT 是基于 C++的，这也意味着开发者可以充分发挥 C++的优势，直接使用 C++进行界面设计和逻辑实现，避免了语言切换和学习成本。

此外，QT 是一个广泛应用的开源框架，拥有强大的社区支持。开发者可以从社区中获取大量文档、教程和解决方案，这有助于提高开发效率并降低开发过程中的困难。QT 还提供了丰富的功能和工具，如可视化设计工具 QT Designer、国际化支持以及对数据库的集成，使得开发者能够更加便捷地构建出功能强大、易用的用户界面。

综合而言，QT 作为前端界面的选择不仅基于其卓越的跨平台性和图形库，还考虑到其与 C++的天然集成、开源社区支持和丰富的功能工具，使得它成为开发数字图像处理系统的理想框架,能够有效满足系统设计和用户界面的需求。

2.2 后端 C++技术

选择 C++作为后端开发语言的决策基于多个关键因素:

首先，C++以其卓越的性能而著称，直接操作内存并提供对硬件的细粒度控制，使其成为处理大规模数据和高计算复杂性任务的理想选择，尤其在数字图像处理的算法实现中。

其次，C++支持面向对象编程，这有助于构建清晰、可维护且可扩展的后端系统。通过使用类和对象，能够更好地组织代码结构，提高代码的可读性和复用性，为系统的长期维护和拓展奠定坚实基础。

C++在系统级编程和底层硬件操作方面表现出色，适用于直接与硬件进行交互的系统开发。在数字图像处理中，需要高效地操作像素数据、进行内存管理等底层任务，C++的强大功能使其成为处理这些任务的首选语言。

广泛的应用领域也是选择 C++的原因之一。其被广泛用于系统开发、游戏开发、嵌入式系统等领域，这使得在 C++上开发的后端系统可以受益于丰富的资源和工具，同时享有广泛的社区支持。

最后，C++具有良好的平台独立性和与其他语言的互操作性，这有助于确保后端系统在不同操作系统上的运行稳定性,同时能够集成其他语言编写的模块，充分利用各语言的优势，提高开发效率。

在数字图像处理系统中，使用 C++作为后端开发语言有助于实现高性能的图像处理算法，同时保障系统的稳健性和可维护性，从而满足复杂图像处理任务的需求。

2.3 运行环境

本数字图像处理系统暂时设计在 Windows 平台上运行。

Windows 平台上，因为选择的 QT 开发版本是 6.4.2,所以只支持 Windows 10 及以上版本。在这一环境下，系统能够充分利用 Windows 操作系统的特性和功能，确保用户在 Windows 系统上获得良好的体验。

为了实现系统的开发和构建，本人使用主流的 C++开发环境，Microsoft Visual Studio Code 实现核心功能,QT 实现界面开发。通过选用适用于 Windows 的编译器 MSVC，本人能够确保系统的源代码能够顺利编译，并在 Windows 平台上正确运行。

系统的图形用户界面采用 QT 框架,通过 QT Creator 进行可视化设计。这不仅有助于提高界面设计的效率,同时也确保了在 Windows 环境下呈现出一致的外观和用户交互体验。

在考虑系统的依赖关系时，本人取消对外部库的依赖，依赖主要集中在标准 C++库。

在用户权限管理方面，系统会合理处理对文件系统、图像数据等资源的访问控制，确保用户在系统范围内获得适当的权限，以保障系统的安全运行。

第 3 章　系统分析

3.1 可行性分析

时机可行性：数字图像处理的项目选择顺应了时代的发展。数字图像处理在当今社会中具有重要性。因此，开发一个不依赖第三方库的数字图像处理程序是与时代潮流契合的。

经济可行性:开发一个不依赖第三方库的数字图像处理程序相对成本较低，因为不需要购买额外的许可或支付第三方库的费用。这降低了经济成本。

操作可行性：项目操作相对直观，因为数字图像处理程序可以被设计成具有用户友好的界面，使得图像处理的操作一目了然，用户能够轻松进行各种图像处理任务。

实现可行性：软件方面，本系统采用的是 C++进行开发，编辑工具使用的是 QTCreator/vscode，项目框架用的是 QT,运行环境是 Windows。

通过分析现有工具与环境，根据所学知识，可以实现系统开发。

3.2 需求分析

3.2.1 用户需求分析

1.易用性:用户界面应简单直观,不需要用户具备专业图像处理知识，即可轻松操作。

2.可定制性：提供足够的参数调整选项，以便用户根据具体需求定制图像处理流程。

3.2.2 功能需求分析

1.图像获取与加载:设计图像获取模块，支持从本地文件系统加载图像数据。此模块负责确保系统能够有效地获取图像。

2.图像处理:开发各种图像算法，如均值滤波、中值滤波、高斯滤波、边缘检测(Sobel)、最临近插值、双立方插值等，以及图像增强算法，如亮度、对比度、饱和度调整等。这些算法可以通过用户界面调整参数，以适应不同的图像处理需求。

3.用户界面设计:使用 QT 框架设计直观的图形用户界面,包括图像展示区域、操作按钮和参数调整控件。用户可以通过界面上传图像、选择算法、调整参数并查看处理结果。

4.性能优化:对系统进行性能优化，包括多线程处理，以确保能够保持高效的性能。

5.用户可以保存处理后的图像，并选择导出目录。

第 4 章　系统设计

4.1 主要功能流程图

主要功能流程图如图 4-1 所示：

用户双击界面进行图片选择操作。首先系统会判断用户是否选择了图像文件，如果未进行选择，或者选择的不是 BMP 文件，则弹出对话框要求用户重新选择。

当用户正常选择 BMP 文件之后，就可以进入图像处理界面对图像进行操作。

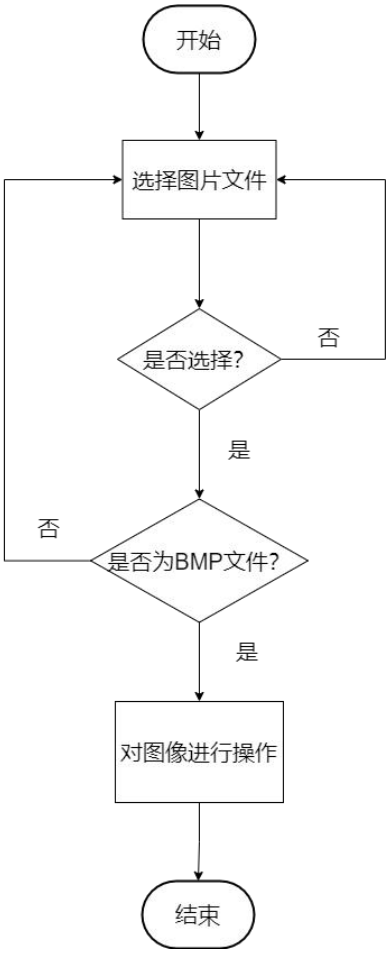

图 4-1 程序主要流程图

4.2 系统界面设计

利用 QTCreator 自带的 UI 设计器进行界面设计，如下图 4-2 所示为 UI 设计器的控件盒子，通过控件盒子进行界面布局。例如，拖放 Push Button 部件到界面，然后设置相对应的槽函数，即可实现按钮功能；拖放 Horizontal Spacer 可以放置一个水平拉伸条。

利用这些控件组合，就可以设计成一个基本的程序界面。

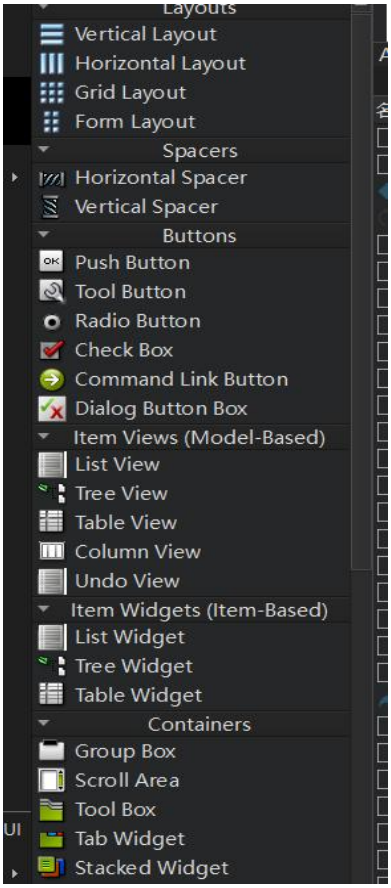

图 4-2 QTCreator 控件

第 5 章　系统原理介绍与系统算法实现

5.1 系统原理介绍

为什么系统选用 BMP 图像进行处理？

1.简单结构:BMP（Bitmap）图像格式具有相对简单的文件结构，它是一种无压缩的位图格式，每个像素都直接映射到图像文件中。这样的简单结构使得读取和处理 BMP 图像的算法相对容易实现。

2.无损压缩:BMP 是一种无损图像格式，不进行图像数据的压缩。在数字图像处理系统中，需要在图像处理的各个阶段观察图像数据的变化，而无损格式能够确保每个处理步骤都不会损失图像质量。

3.广泛支持:BMP 是 Windows 操作系统中常见的图像格式，得到了广泛的支持。它可以被几乎所有的图像处理软件和库所读取和写入，这使得系统更具通用性，方便用户在其他软件中查看处理后的图像。

4.易于学习:对于学习数字图像处理的初学者来说，BMP 格式相对容易理解，因为它的文件结构直观，不涉及复杂的压缩算法。通过对 BMP 格式的处理，学习者可以更好地理解图像处理的基本原理和算法。

5.灵活性:虽然 BMP 是无损格式，但它仍然支持 24 位真彩色图像，提供了足够的灵活性，以适应各种图像处理任务。此外，BMP 格式也可以存储灰度图像和索引颜色图像。

总之，BMP 够被多种 Windows 应用程序所支持，它是 Windows 操作系统中的标准图像文件格式，特点是包含的图像信息较丰富几乎不进行压缩。

因此，开发此系统之前，需要了解 BMP 的文件结构。

如下图 5-1 所示,为 BMP 文件结构:

0~1 字节表示 BMP 文件；第 2~5 字节用小端模式表示这个 BMP 文件的大小；10~13 字节表示图片存储的位置，18~21 字节表示图像宽度；22~25 字节表示图像高度……

图 5-1 BMP 文件结构

了解了 BMP 文件结构之后，就可以进行 BMP 结构的代码编写了。

根据 BMP 文件结构，编写出对应的类 BMP 和 BMPInfo，这两个类是整个系统得以运行的核心代码。

另外，在 C++中，默认的类成员的对齐方式是按照特定的规则进行的，这个规则是由编译器和平台决定的。

大多数情况下，编译器会根据平台的要求选择一个合适的默认对齐方式。

这种默认的对齐方式可以在优化内存访问的性能，因为对齐的数据通常更容易被处理。

但是，因为处理的是 BMP 图像格式，图像数据在内存中的对齐方式需要严格按照 1 字节对齐，所以需要使用 #pragma pack 显式地指定类的对齐方式。

同时，在程序中，会经常出现 reinterpret_cast<T>和 static_cast。

reinterpret_cast<T>表示转换为特定类型（T）的指针。

static_cast 是一种类型转换的安全方式，它可以在不丢失精度的情况下进行数据类型的转换。

在 C++中，这是一种强制类型转换，通常用于处理底层的数据表示，如文件 I/O 中的二进制数据。

有了这些基本头文件信息以及前置操作之后，就可以加载并且读取 BMP 图片信息到内存中了。

定义完 BMP 文件类之后，需要定义一个 MyValue 类，这个类的对象存储的是 BMP 文件对象。

另外还需要定义一个 MyFunction 类，作为函数调用类，这个类包含

了 BMP 文件的读取与保存以及设置 BMP 信息头函数,调用这个类函数，可以把图像数据写入内存中，以便内存数据用于之后的功能函数。

BMP 文件读取的流程：

1.使用 C++自带的文件处理类，以二进制方式读取传入的文件，使用 reintpret_cast<char*>逐字节读取数据，存入 BMP 类中。

2.根据 BMP 类中的 fileType 变量，来判断文件是否为 BMP 文件，判断的条件是 fileType 是否为 BMP 文件对应的 ASCII 码 0x4D42，0x4D42 是 BMP 文件类型标识符，对应于字符"M"和"B"。

3.读取成功并且读入类型是 BMP 文件后，使用 reintpret_cast<char*>确保逐字节读取数据，存入到 BMPInfo 类中，把图像文件的所有信息存入内存中。

4.根据 BMP 文件特性，设置对应的偏移量，同样，使用 reintpret_cast<char*>确保逐字节读取图像 RGB 像素信息。

5.关闭文件。

BMP 文件保存流程：

1.使用 C++自带的文件处理类，以二进制方式写入文件。

2.同样使用 reintpret_cast<char*>确保逐字节写入数据。

3.指针跳转到偏移量位置，写入 RGB 像素信息。

4.关闭文件，文件保存成功。

设置 BMP 文件信息流程：

1.读取 BMP 与 BMPInfo 对象，以及宽高，每像素位数。

2.根据输入的信息，设置新的 BMP 文件头与信息头。

MyFunction 对象可以在外部其他 CPP 文件中被调用。

5.2 系统算法实现

5.2.1 灰度图转换

灰度图转换是整个程序中最为简单的代码,其原理也是非常简单：

读入图像像素信息，通过循环获取到 R、G、B 各个通道的值，然后使用公式：

$$Gray = 0.299 * R + 0.587 * G + 0.114 * B \qquad (5\text{-}1)$$

计算灰度，最后应用到图像每一个通道中即可，下图 5-2 是此过程的流程图。

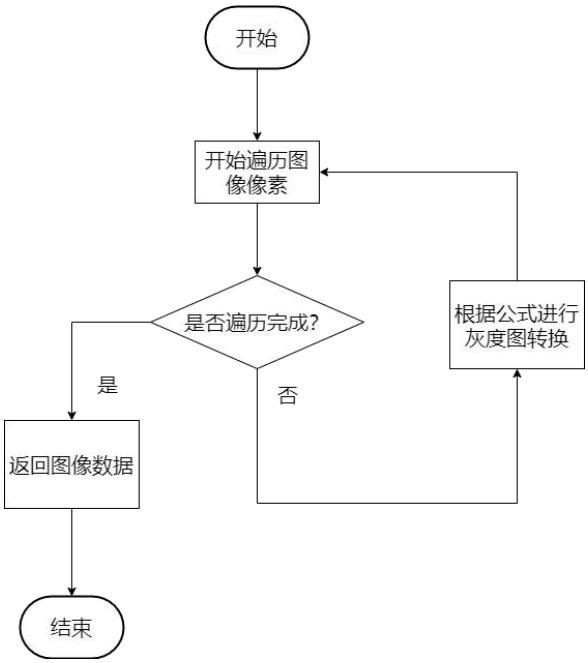

图 5-2 灰度图转换流程图

5.2.2 自动对比度

自动对比度则需要计算两个额外的值--平均值和标准差。

在自动对比度调整中，通过计算平均值和标准差，可以根据图像的整体统计特性来自动调整图像的对比度，使得图像更具视觉效果。

如果图像的像素值分布范围较窄，标准差较小，自动对比度调整可以通过扩展像素值范围来增强图像对比度。

如果像素值范围较宽，标准差较大，可以通过缩小像素值范围来保持图像细节。

这样的处理可以改善图像的质量和可视化效果。

拥有了上述的前置操作，那么接下来就是应用自动对比度了。

在自动对比度的代码中，有这样一行代码：

```
double factor = 128.0 / standard;
```

这行代码是为了计算缩放因子,这个缩放因子的目的是使增强后的像素值范围适应图像的动态范围，并且 128 是被选为中间亮度值，以便在增强中保留图像的细节

接下来是一个 for 循环。

for 循环中部分代码的具体解释如下:

(imageData[i + x] - aver):计算当前像素的 x 通道值相对于整个图像平均值的偏离程度:如果结果为正，表示该像素较亮；如果为负，表示较暗。

factor * (imageData[i + x] - aver):使用缩放因子 factor 对偏

离程度进行调整。这个调整会放大或缩小偏离程度，可以更好地映射到新的对比度范围。

factor * (imageData[i + x] - aver) + 128:最终将调整后的偏离程度加上中间亮度值 128，以确保图像的中间亮度得到保持。为了防止图像过度变暗或过度变亮。

std::max(0, std::min(255, ...)):使用 std::max 和 std::min 函数确保调整后的像素值在合理的范围内(0-255)，超过这个范围的值将被截断。

下图 4-3 为此过程的流程图。

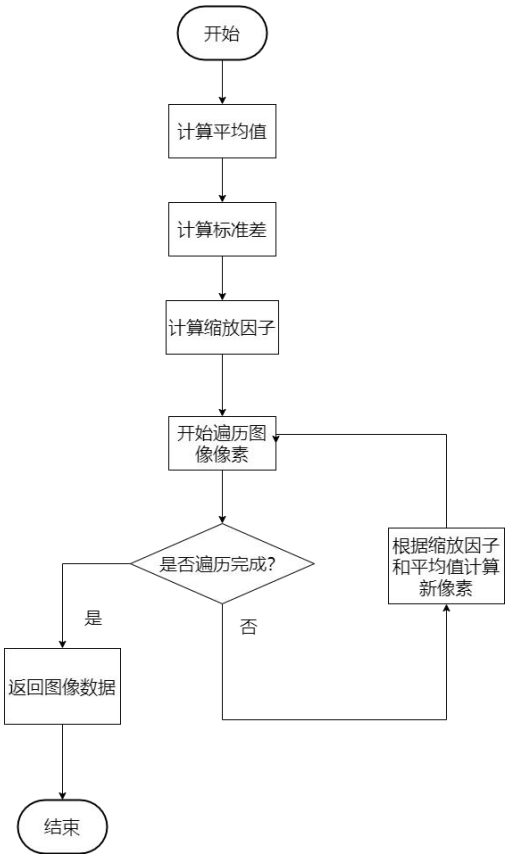

图 4-3 自动对比度流程图

5.2.3 均值模糊

均值模糊,均值模糊需要计算每个像素周围临近像素的平均值,最后取用该平均值代替原始像素值,下图 5-4 为均值模糊原理图[4]:

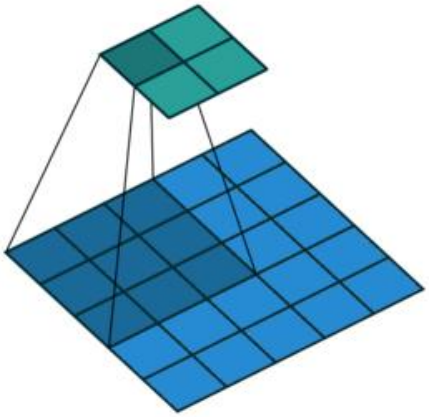

图 5-4 均值模糊原理

在代码中，函数接受外部传入的图像像素数据、宽、高。

接着根据均值模糊原理,使用两个嵌套 for 循环来遍历图像中除了边界之外的每一个像素。

随后在 for 循环中，通过计算索引，获取当前像素位置周围的 8 个像素的值。

最后一步执行计算平均值的操作，并将结果更新到原始图像数据中。

5.2.4 高斯模糊

首先需要高斯函数的公式,高斯公式如下:

$$G(X,Y)=\frac{1}{2\pi\sigma^{2}}e^{-(x^{2}+y^{2})/2\sigma^{2}} \qquad (5\text{-}2)$$

将这个公式转换为代码。

有了高斯公式之后,就需要获得权重矩阵,计算模糊值:

假定中心点的坐标是（0,0），那么距离它最近的 8 个点的坐标如下图 5-5 所示:

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

图 5-5

更远的点以此类推。

为了计算权重矩阵，需要设定 σ 的值。假定 σ =1.5，则模糊半径为 1 的权重矩阵如下图 5-6 所示（把坐标值带入高斯公式）：

0.0453542	0.0566406	0.0453542
0.0566406	0.0707355	0.0566406
0.0453542	0.0566406	0.0453542

图 5-6 权重矩阵

这 9 个点的权重总和等于 0.4787147，如果只计算这 9 个点的加权平均，还必须让它们的权重之和等于 1，因此上面 9 个值还要分别除以 0.4787147，得到最终的权重矩阵。如下图 5-7 所示。

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

图 5-7 最终权重矩阵

有了权重矩阵，就可以计算高斯模糊的值了。

假设现有 9 个像素点，灰度值（0-255），如下图 5-8 所示。

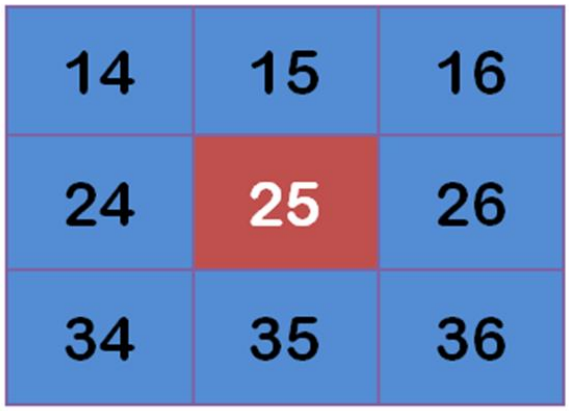

图 5-8

每个点乘以自己的权重值，如下图 5-9 所示。

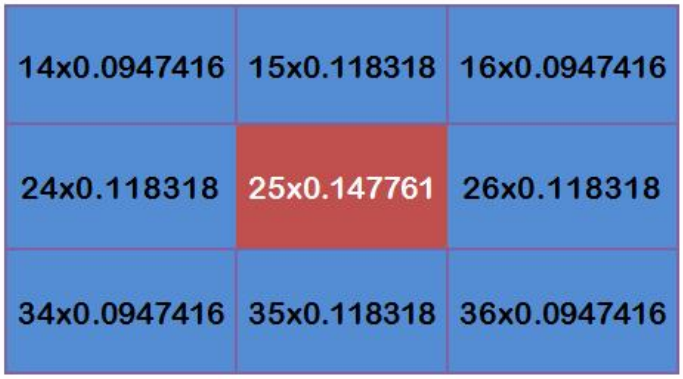

图 5-9

得到如下图 5-10 的结果：

图 5-10

将这 9 个值加起来，就是中心点的高斯模糊的值[4]。

对所有点重复这个过程，就得到了高斯模糊后的图像。对于彩色图片来说，则需要对 RGB 三个通道分别做高斯模糊。

高斯模糊功能函数解释：

1.函数接收一个表示图像像素数据的 std::vector<uint8_t> 类型的参数 imageData,以及图像的宽度、高度和高斯核函数的标准差 sigma.

2.使用两个嵌套循环遍历图像中的每个像素。

3.在嵌套循环中，对于每个像素，计算其周围 3x3 的邻域内的加权平均值（选择 5x5 的高斯核通常是因为它提供了合理的平滑效果，同时保留了边缘特征，并且在计算效率上也相对较高）。

4.对于邻域内的每个有效像素,根据高斯核函数的权重计算加权平均值。

5. 将计算得到的加权平均值更新到原始图像数据中。

此过程的流程图如下图 5-11 所示。

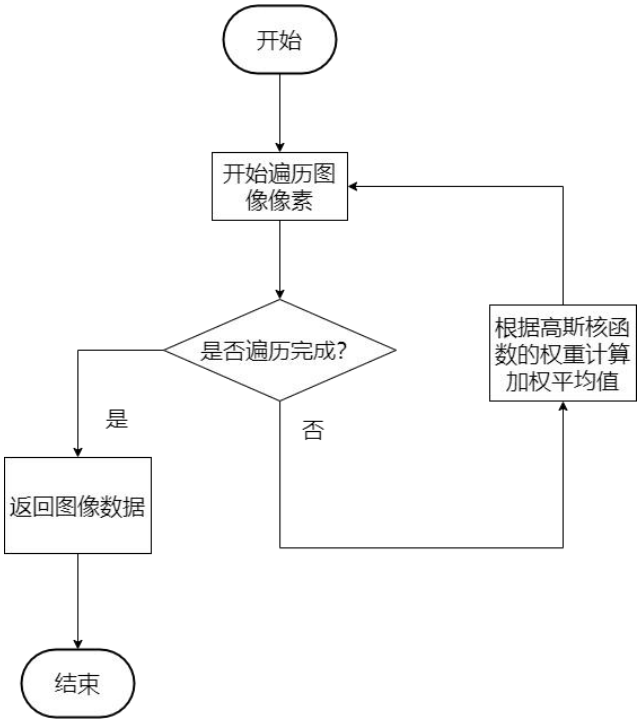

图 5-11 高斯模糊流程图

5.2.5 中值模糊

首先需要获取中间值。

获取中间值函数接受一个窗口（一维向量）作为参数，对窗口内的像素值进行排序，并返回中间值。

中间值是排序后位于中间位置的像素值,这样可以确保在窗口内取得中值。

以下函数为中值模糊的代码解释:

1.该函数接受原始图像数据、图像宽度和图像高度作为参数，对图像进行中值模糊处理。

2.使用嵌套 for 循环遍历像素。

3.在图像的内部区域滑动 3x3 的窗口(这是因为 3x3 窗口足够小以考虑每个像素周围的邻域，同时又足够大以提供中等程度的平滑效果)，每次取窗口内像素值，通过 CalculateMedian 函数计算中值，并将中值赋值给窗口中心像素。

流程图如下图 5-12 所示。

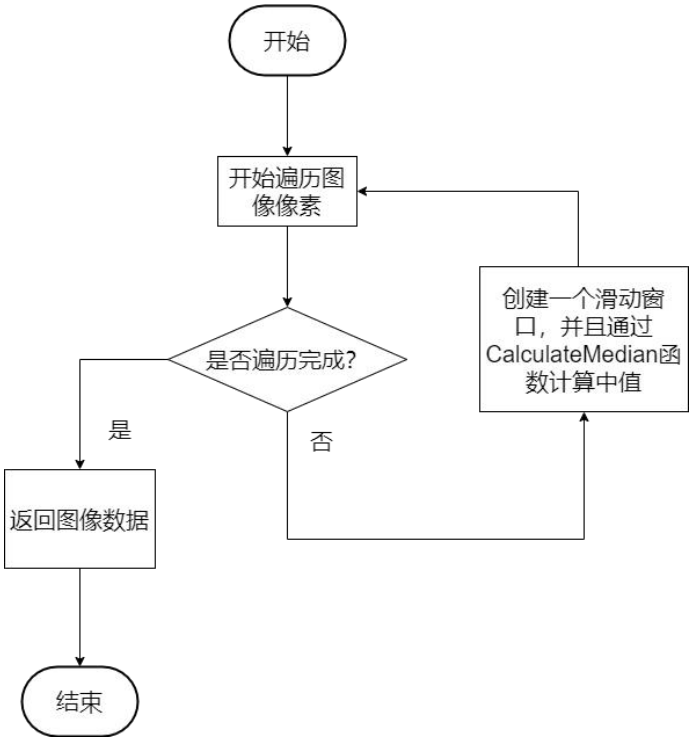

图 5-12 中值模糊流程图

5.2.6 图层混合

定义了四种混合模式,并且存入 emun 枚举类型,以便后续进行选择。

接下来就需要传入两张 BMP 图片,并且分别获取像素信息。

代码流程：

1.输入两张图片并且判断输入的两张图片大小是否相同。

2.通过一个循环，获取原图的 RGB 值与背景图片的 RGB 值：

在代码中，可以看到，原始图片的 RGB 值是通过取地址(&)方式存入变量中的,这意味着对这些引用进行的任何更改都将影响到原图像素数据本身；

背景图片的 RGB 值则是直接定义一个变量进行存储,这意味着对这些变量的任何更改都不会影响到原始背景图像数据。

这种设计选择是因为在混合过程中,通常需要修改原图像素的值以反映混合的效果。

通过使用引用，可以直接在原始图像数据上进行修改，而不必通过索引和复制的方式。

背景像素通常是从另一张图像加载而来，不需要直接修改其值，因此没有使用引用。

3.调用图层混合核心函数。

图层混合核心函数使用的公式解释如下：

正常模式

在正常模式下,混合图像的颜色以及不透明度与背景图像的颜色相结合。这种模式使用以下线性插值公式来混合两个颜色。

公式见下:

$$\text{Result} = (1-\alpha) \times Dest + \alpha \times Src \qquad (5\text{-}3)$$

其中：

ResultResult 是混合后的颜色。

DestDest 是背景图像的颜色。

SrcSrc 是混合图像的颜色。

α 是混合图像的不透明度（0.0 表示完全透明，1.0 表示完全不透明）

在正常模式下，混合程度由不透明度值 α 控制。如果 α 为 0，则完全采用背景图像的颜色，而如果α 为 1，则完全采用混合图像的颜色。在这之间的值会以线性插值的方式混合两者的颜色。

正片叠底

在正片叠底模式下，混合图像的颜色与背景图像的颜色相乘，得到新的混合颜色。

这种混合模式通常用于创建阴影或模拟颜色的遮罩效果。

公式见下:

$$\text{Result} = \left(\frac{Dest \times Src}{255} \right) \qquad (5\text{-}4)$$

在正片叠底模式下，每个颜色通道都会分别相乘，然后除以 255 来保证结果在合理的颜色范围内。

这种模式的特点是深色部分的效果更为明显，而亮色部分相对较浅。

滤色

在滤色模式下,混合图像的颜色与背景图像的颜色进行互补色的像素值相乘，然后除以 255 得到最终的混合颜色。

公式见下:

$$\text{Result} = \left(\frac{(255 - Dest) \times (255 - Src)}{255} \right) \tag{5-5}$$

在滤色模式下，较亮的颜色部分将会被保留，而较暗的颜色将会变得更加透明，产生柔和的混合效果。

这种模式适合用于创建高光和柔和的光影效果。

叠加

在叠加模式下，图像的基色颜色与混合色颜色进行混合，得到新的混合颜色。

公式见下：

如果基色颜色较暗(R、G、B<128)

$$\text{Result} = \left(\frac{2 \times \text{Dest} \times \text{Src}}{255} \right) \tag{5-6}$$

如果基色颜色较亮(R、G、B<128)

$$\text{Result} = \left(255 - \frac{2 \times (255 - \text{Dest}) \times (255 - \text{Src})}{255} \right) \tag{5-7}$$

此过程流程图如下图 5-13 所示。

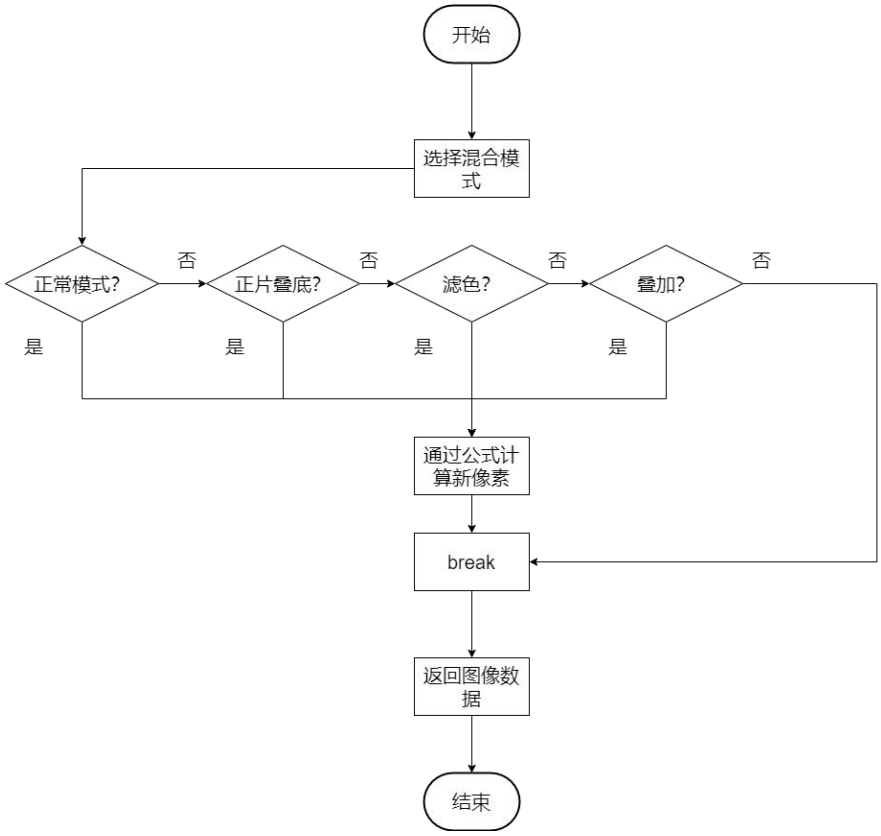

图 5-13 图层混合流程图

5.2.7 色彩映射

色彩映射（Color Mapping）是一种将图像的灰度值映射到颜色空间中的过程。在数字图像处理中，原始图像通常以灰度图的形式存在，即每个像素的亮度由一个灰度值表示。为了更直观地显示图像或突出其中的特定信息，可以通过色彩映射将灰度值映射到彩色空间，从而生成一幅彩色图像。

色彩映射的基本思想是为不同的灰度值分配不同的颜色，这样可以通过颜色的变化来表达图像中灰度的变化。常见的应用包括将地图高度数据映射为彩虹色，或将医学图像中的不同密度区域显示为不同颜色。通过色彩映射，图像的视觉信息更加丰富，更容易理解。

假设我们想在地图上显示美国不同地区的温度。我们可以把美国地图上的温度数据叠加为灰度图像——较暗的区域代表较冷的温度，更明亮的区域代表较热的区域。这样的表现不仅令人难以置信，而且代表了两个重要的原因。首先，人类视觉系统没有被优化来测量灰度强度的微小变化。我们能更好地感知颜色的变化。第二，我们用不同的颜色代表不同的意思。用蓝色和较温暖的温度用红色表示较冷的温度更有意义[6]，如下图 5-14 所示：

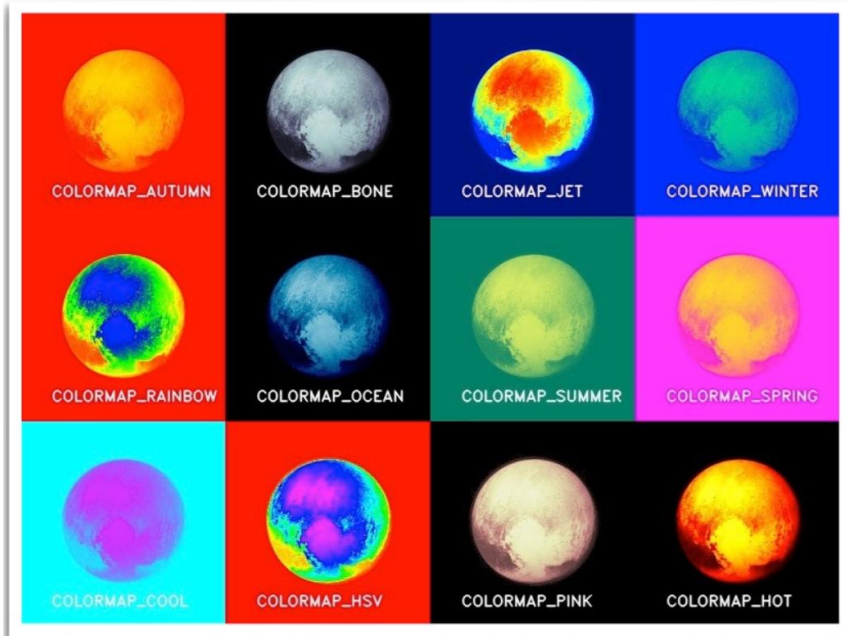

图 5-14 色彩映射的应用

色彩映射的原理如下：

首先将图像转换为灰度图，因为灰度图是一种单通道图像，每个像素的灰度值表示了其亮度，而不涉及彩色信息。

在进行颜色映射时，通过对灰度值的处理和映射，可以更方便地确定

每个像素最终的伪彩色。

这样做的目的是为了简化颜色映射的计算。

在颜色映射中,通常是基于灰度值来决定映射到伪彩色图中的具体颜色[6]。

将图像转换为灰度图后，每个像素只有一个灰度值，简化了颜色映射的计算，使得映射的关系更为直观。

接下来需要定义一个色彩映射表。

在图像处理中，颜色映射表通常用于将灰度图像映射到伪彩色图像，以增强图像的可视效果。

映射的方式通常是通过将灰度值映射到颜色映射表中的对应位置,以实现对图像的重新上色[6]。

定义完图像映射表，下一步就可以进行色彩映射了。

1. 色彩映射函数接受两个参数：图像数据和色彩映射表；

2. 首先需要计算颜色映射表中颜色的数量，因为图像是 RGB 3 通道图像，所以需要先计算出颜色的数量(在颜色映射中，计算颜色数量的目的是为了确定在映射过程中如何将灰度值映射到具体的伪彩色)。

3. 循环遍历每一个像素。

4. 将归一化后的灰度值乘以颜色映射表的长度（颜色数量），得到一个索引值。这个索引值表示了应该从颜色映射表中选择的颜色。为了确保索引值在合理范围内，使用了 numColors - 1。

5. 使用计算得到的索引值，从颜色映射表中选择对应的颜色。这个颜色将被应用到图像的每个通道，从而实现颜色映射效果

此过程流程图如下图 5-15 所示。

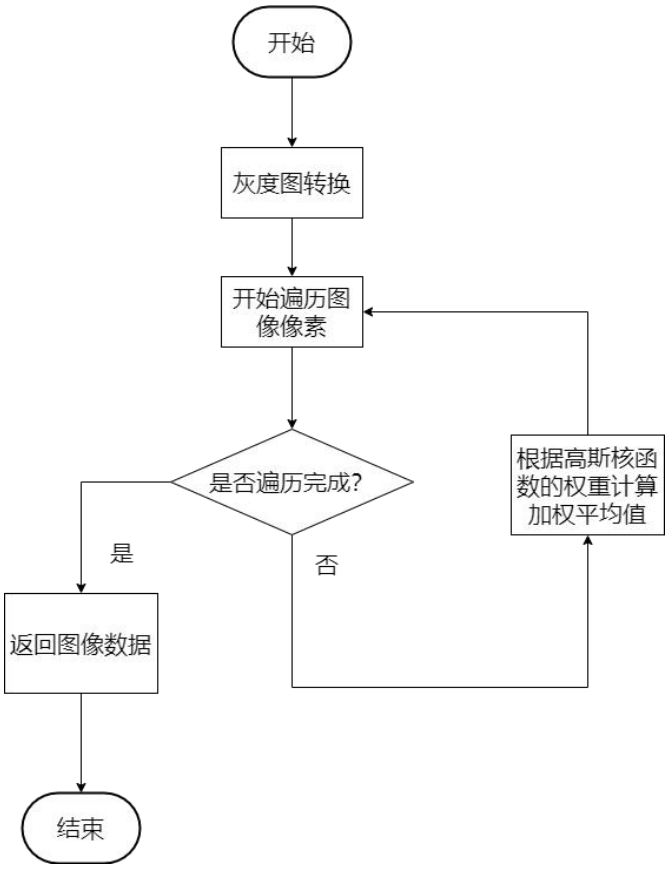

图 5-15 色彩映射流程图

5.2.8 马赛克

FullMosaic 函数接受图像数据、图像宽度、图像高度以及马赛克程度（degree）作为参数。

在函数中，可以看到下列循环条件：

```
for (uint32_t dy = 0; dy < degree && y + dy < height; dy++) {
for (uint32_t dx = 0; dx < degree && x + dx < width; dx++) {
.....
.....
.....
}}
```

这是嵌套的 for 循环，用于遍历图像中的一个马赛克块。具体来说，它用于处理马赛克效果中的每个小块,其中 degree 决定了每个小块的大小。

dy 和 dx 分别代表在当前块中的垂直和水平方向上的偏移。

dy 循环从 0 开始，一直到 degree - 1，控制了垂直方向上的像素遍历。

dx 循环也从 0 开始，一直到 degree - 1，控制了水平方向上的像

素遍历。

dy < degree 保证了在垂直方向上仅遍历 degree 个像素。

y + dy < height 确保在当前块的垂直方向上不会超出图像的高度。

dx < degree 保证了在水平方向上仅遍历 degree 个像素。

x + dx < width 确保在当前块的水平方向上不会超出图像的宽度。

这两个条件的组合确保了循环在处理完整个图像时不会越界。

之后处理每个马赛克块时，计算该块内所有像素的平均颜色，并将整个块的颜色置为平均颜色。

通过对图像进行迭代,将每个指定大小的区域用该区域内像素的平均颜色替代，从而形成全图的马赛克效果。

流程图如下图 5-16 所示。

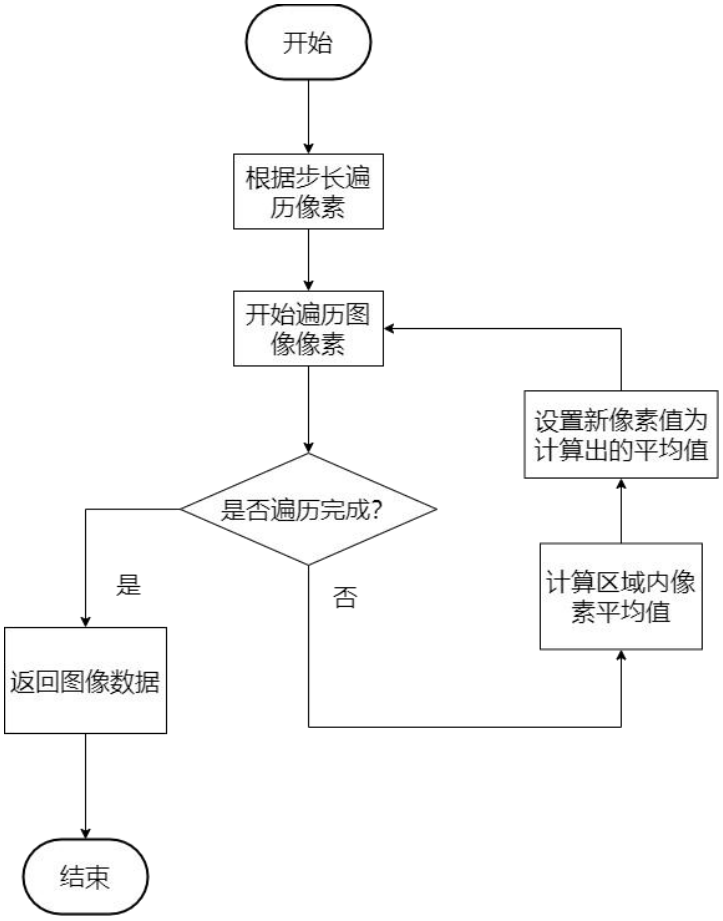

图 5-16 马赛克流程图

5.2.9 插值

接下来介绍三种插值方法：最近邻插值、双线性插值、双立方插值。

最近邻插值

最近邻插值是最简单的一种插值方式，只需要通过映射，将原始图片中的像素值映射到放大（或者缩小）后的图片中的每一个位置上即可，而

不需要通过计算来得到放大后图片中的每一个像素值[8]。

如下图 5-17 所示：

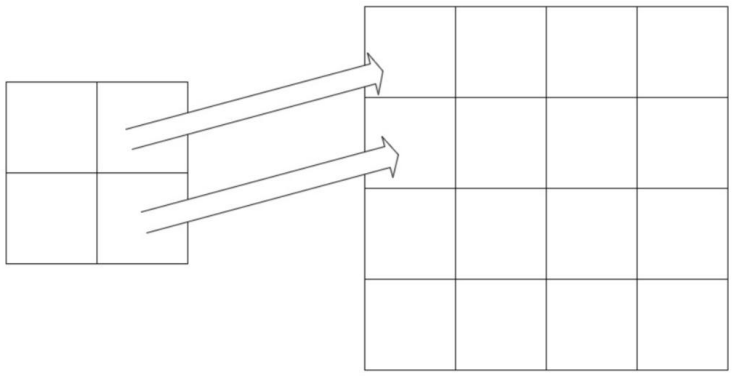

图 5-17 最临近插值

简单的说，就是用把放大图片中的像素值用原始图片中的像素值表示，是放大图片与原始图片内容相似但分辨率增加。

SmallImage 函数：

最近邻插值的思想是在缩小的过程中,每个目标像素对应原图像上最近的像素。

该函数用于实现最近邻插值的图像缩小操作。

接受原始图像数据、原始图像宽度和高度，以及缩小后的目标宽度和高度。

首先需要计算缩小因子 scaleX 和 scaleY。

scaleX 和 scaleY 是水平和垂直方向的缩放因子。

x 和 y 是新图像中的像素坐标。

srcX 和 srcY 是通过缩放因子计算得到的原始图像中最近邻的像素坐标。

代码目的是将新图像中的坐标映射到原始图像中,以便确定最近邻的像素值。

这就是最近邻插值方法的基本原理之一。

然后对目标图像的每个像素,通过最近邻插值从原图像中找到对应的像素值，进行复制。

LargeImage 函数：

该函数用于实现最近邻插值的图像放大操作。

接受原始图像数据、原始图像宽度和高度，以及放大后的目标宽度和高度。

计算放大因子 scaleX 和 scaleY，然后对目标图像的每个像素，通过最近邻插值从原图像中找到对应的像素值进行复制,操作方法与上述缩小代码类似，此处不再赘述。

最近邻插值的思想是在放大的过程中,每个目标像素对应原图像上最近的像素。

此过程如下图 5-18 所示。

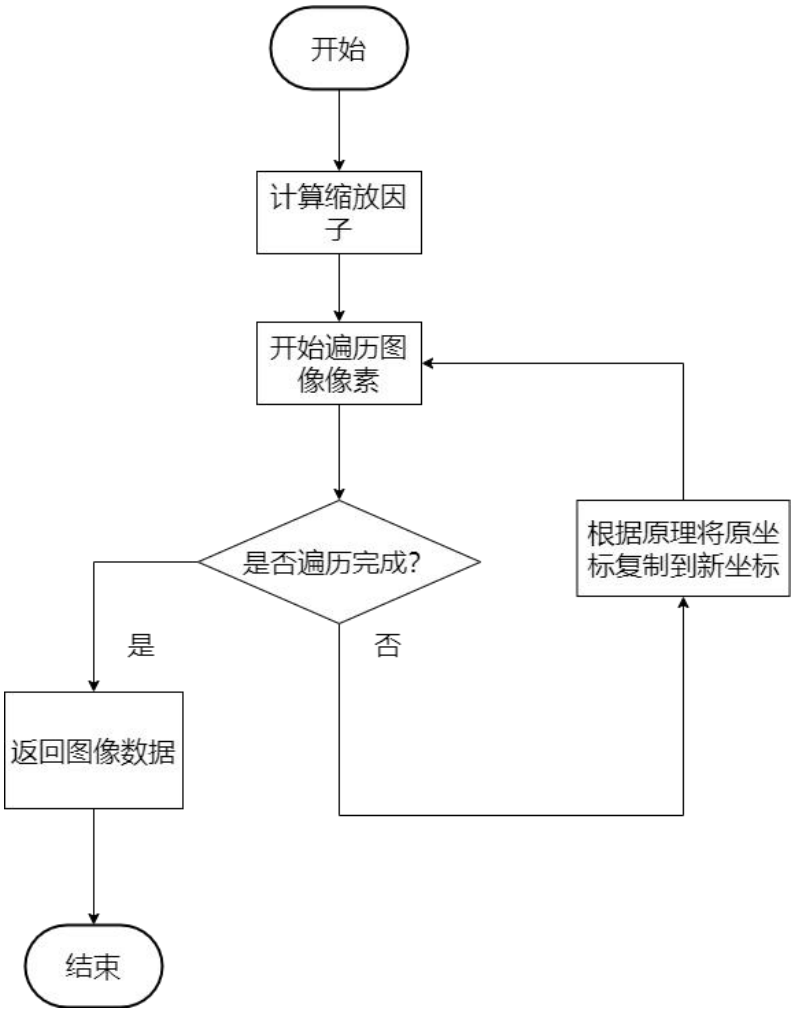

图 5-18 最临近插值流程图

双线性插值

在了解双线性插值之前,首先需要了解单次线性插值。

如下图 5-19 所示。

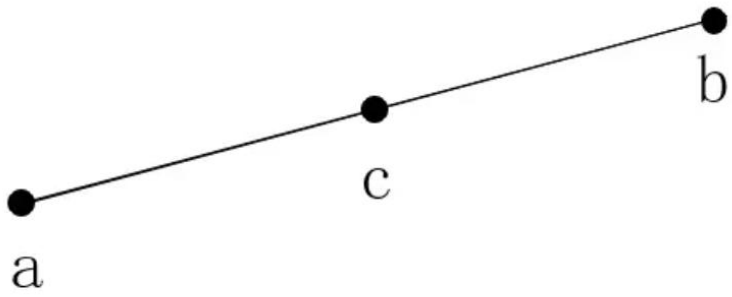

图 5-19 单次线性插值

假设二维空间中有一条直线，其上有两个点 a(x1,y1)，b(x2,y2)，x1、x2 为坐标，y1、y2 为函数值，我们需要通过 c 点的坐标 x 来表示 c 点的函数值 y。因为直线上的函数值是线性变化的，我们只需通过计算 a、c 两点斜率和 a、b 两点的斜率，令二者相等可以得到一个方程[8]，如下所示:

$$\frac{f(x1)-f(x)}{x1-x}=\frac{f(x1)-f(x2)}{x1-x2} \tag{5-8}$$

再次化简得到 c 点函数值:

$$f(x)=\frac{x-x2}{x1-x2}f(x1)+\frac{x1-x}{x1-x2}f(x2) \tag{5-9}$$

所谓双线性插值，就是在两个方向上进行了插值，总共进行了三次插值，如下图 5-20 所示。

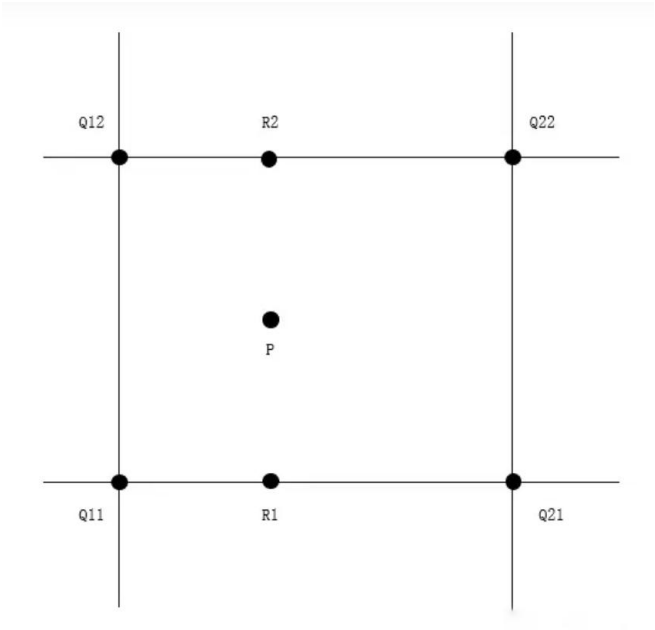

图 5-20 双线性插值

如上图所示，三维空间（x,y 为坐标，第三维为图片的像素值）中，我们需要计算出 P 点的像素值。我们取 P 点邻近的四个点 Q11,Q12,Q21,Q22，并假设在邻近范围内，点的像素值是呈线性变化的。这时我们先在 x 方向上进行单次线性插值，计算出 R1,R2 的像素值，再在 y 方向上进行单次线性插值，求出 P 的像素值。

这里的这些点都是位于原始图像上的，我们只需要找到一个映射公式，将放大图像上的点与 P 点对应即可。当然这里的映射公式就是最近邻插值中的映射公式。

与最近邻插值法不同的是,双线性插值法并没有之间将映射点的像素值作为放大图像的像素值,而是将映射点周围的四个点的加权作为放大图像的像素值[8]。

具体的计算公式如下:

$$\begin{aligned} f(R1) &= \frac{x2-x}{x2-x1} f(Q11) + \frac{x-x1}{x2-x1} f(Q21) \\ f(R2) &= \frac{x2-x}{x2-x1} f(Q12) + \frac{x-x1}{x2-x1} f(Q22) \\ f(P) &= \frac{y2-y}{y2-y1} f(R1) + \frac{y-y1}{y2-y1} f(R2) \end{aligned} \tag{5-10}$$

整合得:

$$\begin{aligned} f(P) = {} & \frac{(x2-x)(y2-y)}{(x2-x1)(y2-y1)} f(Q11) + \frac{(x-x1)(y2-y)}{(x2-x1)(y2-y1)} f(Q21) \\ & + \frac{(x2-x)(y-y1)}{(x2-x1)(y2-y1)} f(Q12) + \frac{(x-x1)(y-y1)}{(x2-x1)(y2-y1)} f(Q22) \end{aligned} \tag{5-11}$$

SmallImage 函数：

该函数用于实现双线性插值的图像缩小操作。

接受原始图像数据、原始图像宽度和高度，以及缩小后的目标宽度和高度。

计算缩小因子 scaleX 和 scaleY，然后对目标图像的每个像素，通过双线性插值从原图像中找到对应的像素值进行复制。

```
auto x1 = static_cast<int32_t>(srcX);
auto x2 = static_cast<int32_t>(x1 + 1);
auto y1 = static_cast<int32_t>(srcY);
auto y2 = static_cast<int32_t>(y1 + 1);
```

上方这段代码通过将浮点坐标转换为整数,得到目标像素在原图中最近的四个像素点的坐标。

```
auto tx = srcX - x1;
auto ty = srcY - y1;
auto w1 = (1.0 - tx) * (1.0 - ty);
auto w2 = tx * (1.0 - ty);
auto w3 = (1.0 - tx) * ty;
auto w4 = tx * ty;
```

上方这段代码计算了目标像素与最近的四个原始像素之间的权重：

srcX 是目标图像中的横坐标,而 x1 是最近邻左上角像素的横坐标。因此，srcX - x1 表示目标像素相对于最近邻左上角像素在 x 方向上的偏移量。这个偏移量用于计算目标像素在 x 方向上相对于最近邻像素的位置。

tx 和 ty 分别是 srcX 和 srcY 相对于最近邻左上角像素的偏移量。这两个值表示了目标像素相对于最近邻像素的位置。

w1、w2、w3 和 w4 是四个最近邻像素的权重。这些权重用于计算目标像素的颜色值，确保在插值计算中，离目标像素越近的原始像素权重越大。

w1 = (1.0 - tx) * (1.0 - ty)：左上角像素的权重，表示 tx 和 ty 对应的偏移量都为零。

w2 = tx * (1.0 - ty)：右上角像素的权重，表示在 x 方向上相对左上角像素有偏移。

w3 = (1.0 - tx) * ty：左下角像素的权重，表示在 y 方向上相对左上角像素有偏移。

w4 = tx * ty：右下角像素的权重，表示在 x 和 y 方向上都相对左上角像素有偏移。

w1、w2、w3 和 w4 分别代表四个最近邻像素的权重，这些权重将在插值计算中用于组合原始像素的颜色值。

双线性插值考虑了目标像素与最近的四个原始像素之间的权重关系，计算了四个权重 w1、w2、w3 和 w4。

最后，根据计算得到的权重，对最近邻的四个原始像素的颜色值进行插值计算，得到目标像素的新颜色值。循环中的 channel 表示颜色通道，分别进行插值计算。

LargeImage 函数：

该函数用于实现双线性插值的图像放大操作。

接受原始图像数据、原始图像宽度和高度，以及放大后的目标宽度和高度。

计算放大因子 scaleX 和 scaleY，然后对目标图像的每个像素，通过双线性插值从原图像中找到对应的像素值进行复制。

双线性插值考虑了目标像素与最近的四个原始像素之间的权重关系，计算了四个权重 w1、w2、w3 和 w4。

流程图如下图 5-21 所示。

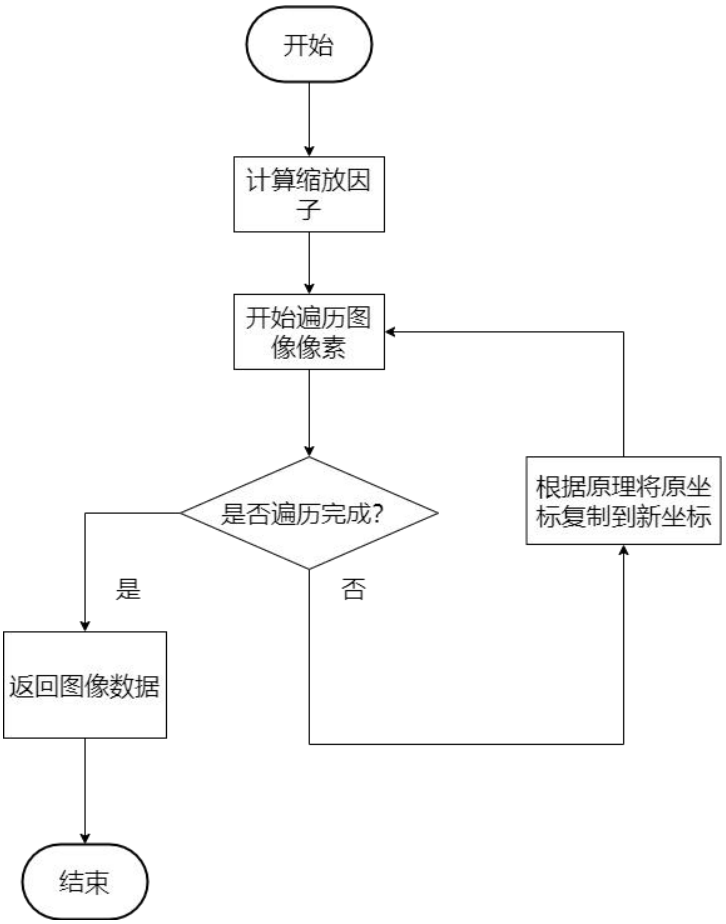

图 5-21 双线性插值流程图

双立方插值:

与双线性插值法相同，该方法也是通过映射，在映射点的邻域内通过加权来得到放大图像中的像素值。不同的是，双三次插值法需要 P 点近邻的 16 个点来加权[8]。如下图 5-22 所示:

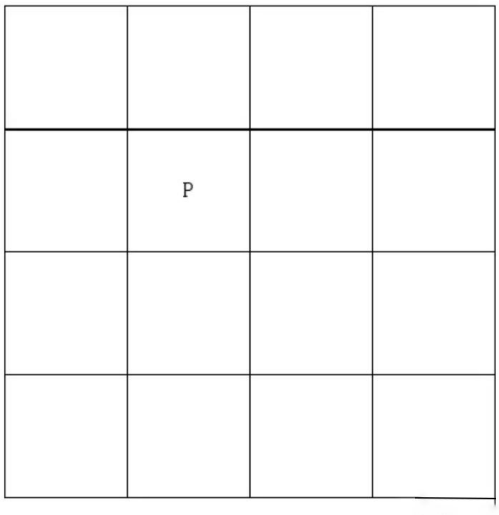

图 5-22 双立方插值

这里我们首先构造一个 BiCubic 函数,它是用来根据近邻点与 P 点的相对位置来计算该点前的权值的一个函数：

$$W(x)=\begin{cases} x=(a+2)|x|^3-(a+3)|x|^2+1 & |x|\le 1 \\ y=a|x|^3-5a|x|^2+8a|x|-4a & 1<|x|<2 \\ z=0 & otherwise \end{cases} \quad (5\text{-}12)$$

这里 a 一般取-0.5。

得到权值后，我们只需要将这 16 个点的像素值加权起来即可，插值计算的公式如下：

$$\mathrm{f(x,y)}=\sum_{i=0}^{3}\sum_{j=0}^{3}f(x_i,y_j)W(x-x_i)W(y-y_j) \quad (5\text{-}13)$$

而关于 P 点的选取则是按照最近邻插值法的映射公式来得到，P 点的最近邻的 16 个点也是按照上图中的相对位置来进行选取[8]。

首先需要一个权重函数：

该函数实现了双立方插值法中的权重计算。

根据插值点距离的绝对值，计算出三种情况下的权重。

以下是放大缩小的函数代码解释:

LargeImage 函数：

该函数用于实现双立方插值的图像放大操作。

首先需要根据目标图像的坐标 (x, y) 计算对应于原图像中的浮点坐标 (srcX, srcY)。

接下来需要对 (srcX, srcY) 进行边界检查，确保它在原图像的有效范围内。这是为了防止插值过程中访问到图像边界外的无效像素。

接下来设置一个用于进行双三次插值的权重矩阵 weights[4][4]。

这个权重矩阵用于在原始图像的周围 16 个像素上执行插值计算。

接下来的这两个循环遍历了一个以目标像素 (x, y) 为中心的 4x4 的区域。

循环开始是从-1 开始的，因为在双三次插值的权重计算中，使用 -1 到 2 的范围是为了考虑到目标像素周围的一个 4x4 区域。

这个区域的中心是目标像素，而 -1 和 2 的范围确保了在水平和垂直方向上都覆盖了相邻的像素。

```
int xi = static_cast<int>(std::floor(srcX)) + i;
int yj = static_cast<int>(std::floor(srcY)) + j;
```

上方代码对于每个循环迭代，计算了原始图像上的相应坐标 (xi, yj)。

```
xi = std::max(0, std::min(xi, width - 1));
```

yj = std::max(0, std::min(yj, height - 1));

上方代码对 (xi, yj) 进行边界检查，确保它在原始图像的有效范围内。

这是为了防止插值过程中访问到图像边界外的无效像素。

float wx = cubicWeight(srcX - (xi + 0.5f));

float wy = cubicWeight(srcY - (yj + 0.5f));

在双三次插值中，srcX 和 srcY 是目标像素在原始图像中的浮点坐标，(xi + 0.5f) 和 (yj + 0.5f) 则是相邻像素在原始图像中的中心坐标。

(xi + 0.5f) 和 (yj + 0.5f) 表示相邻像素的中心，而 (srcX - (xi + 0.5f)) 和 (srcY - (yj + 0.5f)) 则表示目标像素与相邻像素中心之间的相对位置。

总之，这个偏移量是为了计算插值核函数的值，以确定目标像素的最终插值。

这种相对位置的选择是为了计算插值核（权重函数）的值。在双三次插值中，通过将相对位置传递给权重函数，可以获得目标像素与相邻像素之间的插值权重。

使用 cubicWeight 函数计算在 (srcX, srcY) 周围像素位置 (xi, yj) 处的双三次插值权重。这两个权重分别是 wx 和 wy。

这个权重矩阵 weights[4][4] 中的每个元素都代表了在进行插值计算时，对应原始图像上的一个像素的权重。

在进行双三次插值时,这些权重用于对原始图像中的像素进行加权平均，以计算目标图像中每个像素的值。

float interpolatedValue = 0.0f;

上面这一行代码表示在每一个通道(R,G,B)中，初始化插值的值为 0，

接下来的两个循环迭代插值核的每个元素。

首先需要计算当前插值核元素在原始图像中的坐标。

std::floor 用于将浮点坐标向下取整，-1 是因为在权重计算中使用的是以目标像素为中心的相对坐标。

接着两行代码是为了确保计算得到的坐标在原始图像的范围内,防止越界。

最后利用双三次插值权重和原始图像的像素值,计算出当前插值核元素的贡献，并累加到 interpolatedValue 中。

上述这个过程会通过遍历所有插值核元素，计算它们的贡献，并加权

求和，得到目标像素在当前通道的最终插值值。

这整个过程在三个通道上独立进行，因为每个通道都有自己的插值值。

最后：

1. 由于像素值通常在 0 到 255 之间，所以使用 std::max(0.0f, std::min(255.0f, interpolatedValue)) 将插值后的值截取到这个范围内。

2.将截取后的插值像素值转换为 uint8_t 类型，并存储在 resizedImage 中的相应位置。

这步操作确保了最终的插值像素值在合理的范围内，并以 uint8_t 类型保存，以符合图像的存储格式。

SmallImage 函数：

该函数用于实现双立方插值的图像缩小操作。

对目标图像的每个像素，计算出其在原图像中的对应位置，并使用双立方插值法计算插值结果。

对于每个插值位置，计算插值点周围的权重，并根据这些权重和对应位置的像素值计算插值结果。

函数与放大函数大同小异，此处不再赘述。

流程图如下图 4-23 所示。

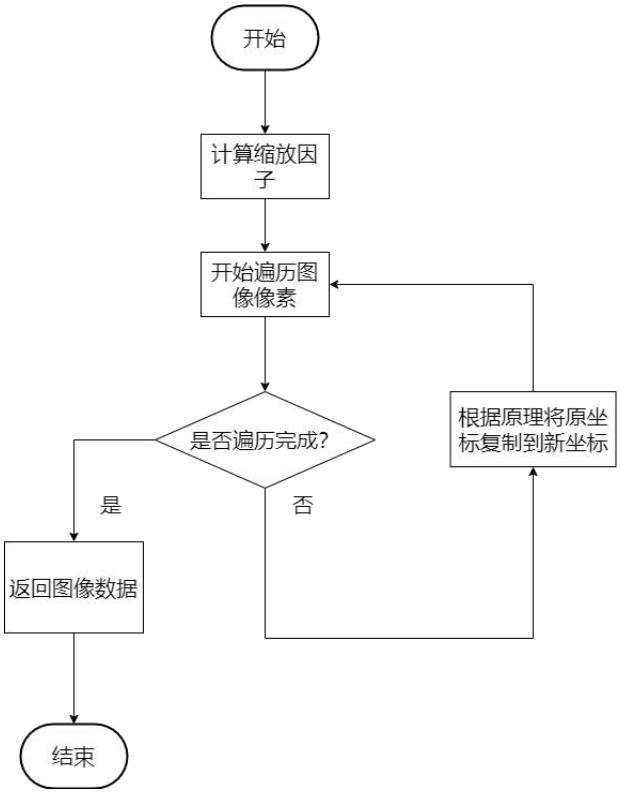

图 4-23 双立方插值流程图

5.2.10 边缘检测

索伯算子（Sobel operator）是图像处理中的算子之一，有时又称为索伯-费尔德曼算子或索伯滤波器，在影像处理及电脑视觉领域中常被用来做边缘检测。在技术上，它是一离散性差分算子，用来运算图像亮度函数的梯度之近似值。在图像的任何一点使用此算子，索伯算子的运算将会产生对应的梯度向量或是其范数。概念上，索伯算子就是一个小且是整数的滤波器对整张影像在水平及垂直方向上做卷积,因此它所需的运算资源相对较少，另一方面，对于影像中的频率变化较高的地方，它所得的梯度之近似值也比较粗糙[9]。

Sobel 算子是一种基于卷积的边缘检测算法，其原理是通过卷积运算来计算图像中每个像素点的梯度值，从而检测出边缘。Sobel 算子分别采用了水平方向和垂直方向的卷积核,用于计算图像中每个像素点在水平和垂直方向上的梯度值。具体来说，Sobel 算子采用如下两个卷积核进行卷积运算[9]：

水平方向如下图 5-24 所示:

```
-1  0  1
-2  0  2
-1  0  1
```

图 5-24 水平方向卷积核

垂直方向如下图 5-25 所示:

```
-1 -2 -1
 0  0  0
 1  2  1
```

图 5-25 垂直方向卷积核

通过水平方向卷积核和垂直方向卷积核，可以定义得到两个数组，这两个数组分别对应水平方向卷积核和垂直方向卷积核,分别用于检测图像的水平边缘和垂直边缘。

接下来就是边缘检测的核心函数。

对于一张图像，我们可以通过对其进行水平和垂直方向上的卷积运算，得到其在每个像素点处的梯度值。

该函数接受图像数据、图像宽度和高度作为参数，并返回进行 Sobel 边缘检测后的图像数据。

sumX 和 sumY 用于存储水平和垂直方向上的梯度。

在循环中，对图像中的每个像素应用 Sobel 算子进行卷积计算，得到

水平方向和垂直方向的梯度。

pixelX 和 pixelY 这两行计算了在 3x3 卷积核内当前迭代的邻近像素的坐标。

当前处理的像素是(x, y)，而 i 和 j 遍历 3x3 卷积核，因此 pixelX 和 pixelY 表示邻近像素的坐标。

```
int pixelIndex = (pixelY * width + pixelX) * 3;
```

上面这一行计算了在 3x3 卷积核内当前迭代的邻近像素在一维图像数据中的索引。因为图像数据通常是三个通道，因此乘以 3。

```
sumX += grayValue * sobelX[j][i];
sumY += grayValue * sobelY[j][i];
```

上面这两行执行卷积操作。通过将邻近像素的灰度值与相应的 Sobel 核的权重相乘，然后将结果累加到 sumX（水平方向）和 sumY（垂直方向）中。

之后计算梯度的绝对值之和，并将其用作像素的边缘强度。

最后使用上一步的边缘强度值用于设置新图像中相应像素的 RGB 通道值，从而呈现边缘检测的结果。

流程图如下图 5-26 所示。

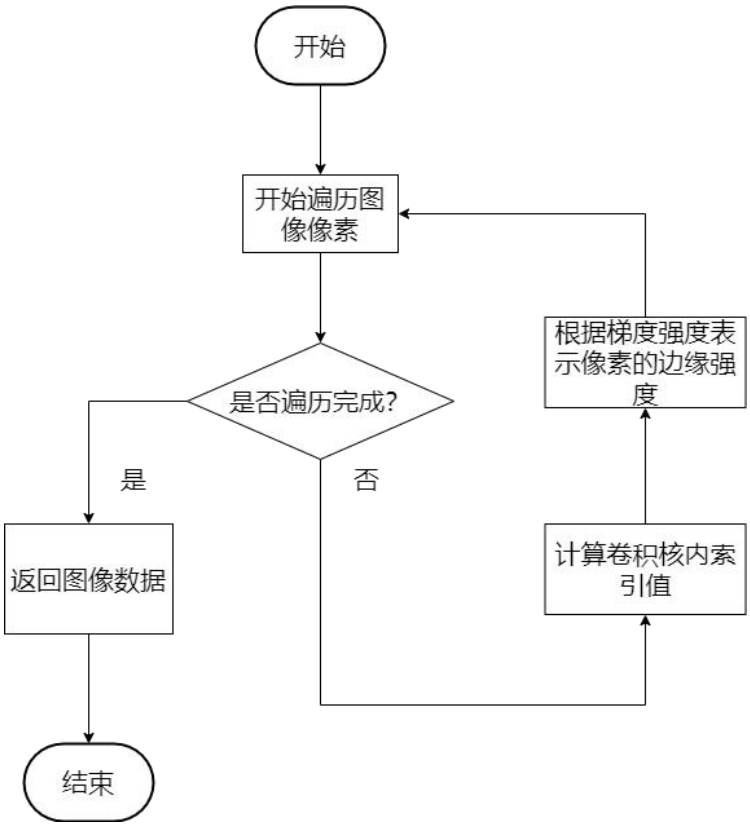

图 5-26 边缘检测流程图

5.2.11 眼部区域液化

此函数接收图像数据，宽度，高度，眼部 x 坐标与眼部 y 坐标，液化半径和液化值。

其中，眼部坐标通过 QT 的鼠标点击事件来获取其 x，y 值，半径和液化值通过 QT 的界面控件输入来获取。

```
double warpAmount = intensity * std::sin(distance / radius * 3.14159265);
```

上面这行代码的作用是根据距离（distance）、半径（radius）和强度（intensity）来计算像素的液化量:

distance: 表示像素到中心点的距离。距离越大，distance 的值越大。

radius: 是液化的半径，即在这个半径范围内的像素会被液化。

intensity: 是液化的强度，用来控制液化的程度。

这行代码先将 distance / radius 的比值乘以 π（π ≈ 3.14159265），然后取正弦函数的值。最后将结果乘以 intensity，得到液化的量 warpAmount。

这样，warpAmount 的值就是根据距离、半径和强度计算出的液化量，用来调整像素的位置，实现液化效果。

```
int x0 = static_cast<int>(newX);
int y0 = static_cast<int>(newY);
int x1 = std::min(x0 + 1, width - 1);
int y1 = std::min(y0 + 1, height - 1);
double dx = newX - x0;
double dy = newY - y0;
```

以上这部分代码是为了进行双线性插。在液化过程中，新的像素坐标 newX 和 newY 是浮点数，而图像的像素坐标是整数。

因此，需要通过双线性插值来计算目标像素的颜色值，以使图像看起来更平滑:

x0 和 y0 是 newX 和 newY 的整数部分，表示目标像素坐标的左上角像素位置。

x1 和 y1 是 x0 和 y0 分别加 1 后的值,表示目标像素坐标的右下角像素位置。这里使用 std::min 函数确保不超出图像边界。

dx 和 dy 是 newX - x0 和 newY - y0，表示目标像素坐标的浮点偏移量。

这样，x0 和 y0 表示左上角像素，x1 和 y1 表示右下角像素，而 dx 和 dy 表示在这个像素坐标中的相对位置。这些信息将用于进行双线性插值，以获取目标像素的颜色值。

接下来就需要实现双线性插值，计算目标像素的颜色值。

在代码中 targetPixelIndex00、targetPixelIndex01、targetPixelIndex10 和 targetPixelIndex11 分别是目标像素周围四个相邻像素的索引。

imageData[targetPixelIndex00 + c] 表示左上角像素在当前颜色通道上的值乘以 (1 - dx) * (1 - dy)。

imageData[targetPixelIndex01 + c] 表示右上角像素在当前颜色通道上的值乘以 dx * (1 - dy)。

imageData[targetPixelIndex10 + c] 表示左下角像素在当前颜色通道上的值乘以 (1 - dx) * dy。

imageData[targetPixelIndex11 + c] 表示右下角像素在当前颜色通道上的值乘以 dx * dy。

这些值通过线性插值相加，然后被转换为无符号 8 位整数，并存储在 imageData[pixelIndex + c] 中，完成了对目标像素颜色的插值。

上述这个过程使得图像在进行液化操作时能够更平滑地改变像素位置。

流程图如下图 5-27 所示。

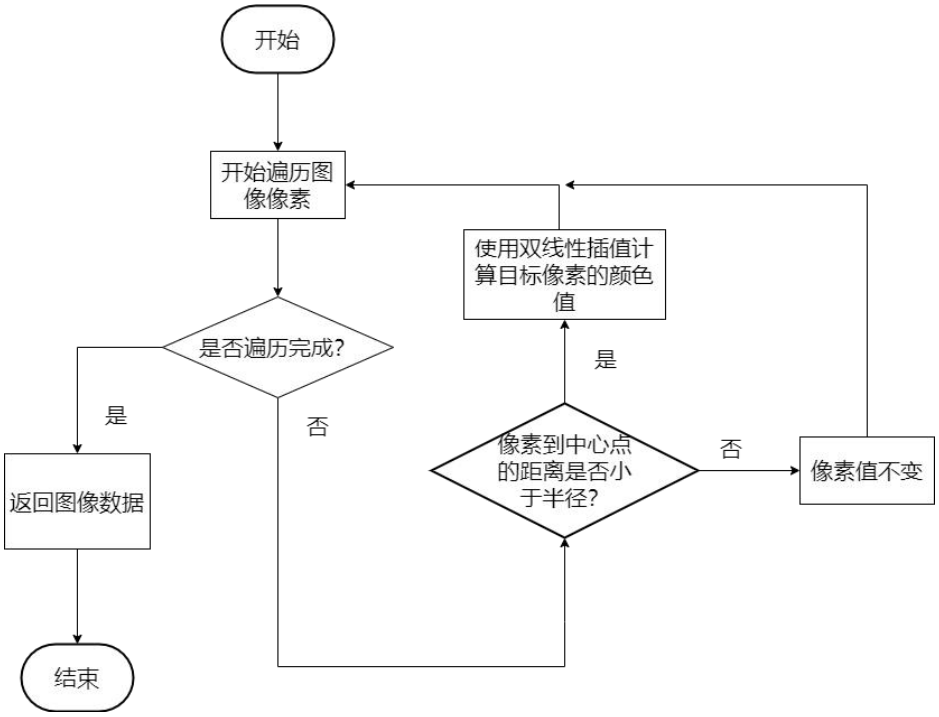

图 5-27 眼部区域液化流程图

5.2.12 鱼眼镜头

研究表明鱼眼相机成像时遵循的模型可以近似为单位球面投影模型。可以将鱼眼相机的成像过程分解成两步：第一步，三维空间点线性地投影到一个球面上，它是一个虚拟的单位球面，它的球心与相机坐标系的原点重合；第二步，单位球面上的点投影到图像平面上，这个过程是非线性的。下图 5-28 表示出了鱼眼相机的成像过程[14]。

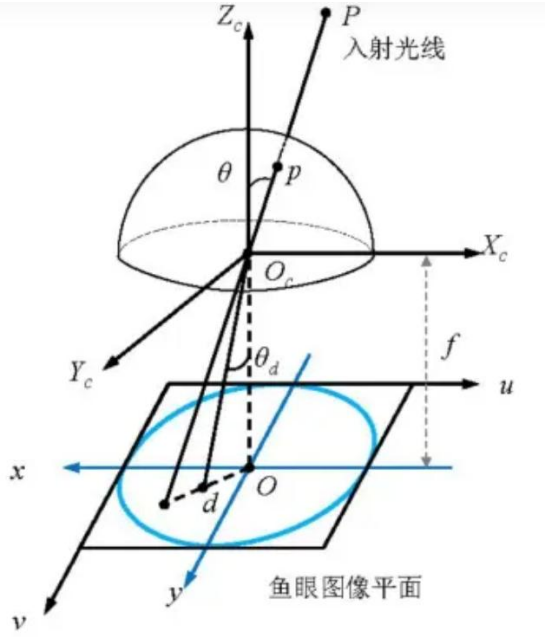

图 5-28 鱼眼相机成像过程

鱼眼函数接收图像数据，宽度，高度：

函数中涉及到的知识点如下:

极坐标变换：

鱼眼效果的实现核心是将直角坐标系下的图像映射到极坐标系下。

对于每个像素，首先计算它相对于图像中心的极坐标角度和距离。

对于在最大半径范围内的像素，应用非线性变换，将距离进行映射，模拟鱼眼效果。

通过极坐标变换，计算新的像素位置。

非线性变换:

normalizedRadius 表示当前像素到中心的距离归一化到范围 [0, 1]。

mappedRadius 利用非线性映射，平方，对归一化距离进行变换，增强鱼眼效果。

双线性插值：

在计算新的像素位置时，使用了双线性插值来获取目标像素的颜色

值。

这是因为新的像素位置通常不是整数，需要通过插值计算得到。

float angle = std::atan2(static_cast<float>(offsetY),static_cast<float>(offsetX));

上面这行代码使用 std::atan2 函数(std::atan2 是 C++ 标准库中的数学函数，用于计算给定的 y 和 x 坐标的反正切值。此代码中 atan2 函数返回两个参数的反正切值，即给定 y 和 x 坐标，它返回的是点(x,y)处的极坐标角度)

计算当前像素相对于图像中心点的极坐标角度。std::atan2 函数返回给定坐标 (y, x) 的极坐标角度，范围是从负 π 到正 π 。

此处，offsetY 和 offsetX 是当前像素相对于图像中心的偏移量，因此 std::atan2(static_cast<float>(offsetY), static_cast<float>(offsetX)) 返回了当前像素的极坐标角度。

在极坐标中，这个角度用于确定鱼眼效果中每个像素的位置。

在接下来的代码中:

normalizedRadius 表示当前像素到图像中心的距离除以最大半径的比例。这个值在范围 [0, 1] 内。

mappedRadius 对 normalizedRadius 进行非线性变换，将距离的变化映射到一个新的值。这里使用了平方操作 (normalizedRadius * normalizedRadius)，这是一个常见的非线性变换，用于增强效果。

angle 是当前像素相对于图像中心点的极坐标角度，通过上述 std::atan2 计算得到。

newX 和 newY 是进行极坐标变换后的新坐标。根据鱼眼效果，通过 mappedRadius、angle 以及原始的图像中心点坐标，使用极坐标公式：

$$\begin{aligned} x' &= x_0 + r \cdot \cos(\theta) \\ y' &= y_0 + r \cdot \sin(\theta) \end{aligned} \tag{5-14}$$

计算出新的坐标值。

x′和 y′是直角坐标系中的新坐标。

x_0 和 y_0 极坐标系的原点(图像中心)。

r 表示极径，是点到原点的距离。

θ 是极角，表示点在极坐标系中与参考方向的夹角。

这两个公式的来源基于三角函数的性质。在直角三角形中 r •cos(θ)

表示点在 x 方向上的投影，r * sin(θ) 表示点在 y 方向上的投影，通过这两个投影，可以得到点在直角坐标系中的坐标。这种表示方式更自然地描述了极坐标系中的点相对于原点的位置。

在鱼眼效果中，通过这个极坐标变换，可以实现将图像中心附近的点保持相对不变，而远离中心的点进行放大，从而产生鱼眼效果。

流程图如下图 5-29 所示。

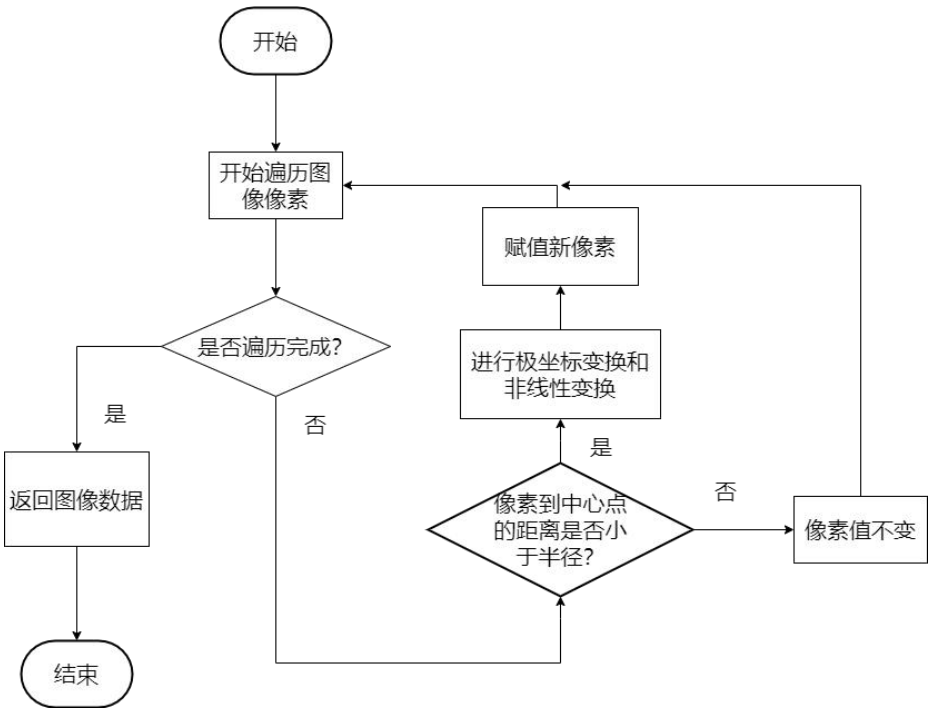

图 5-29 鱼眼镜头流程图

5.2.13 颜色反转

颜色反转函数相对简单。

函数通过接收图像数据,之后通过迭代图像数据的每个像素的三个颜色通道，用 255 减去原始像素，得到的就是反转后的颜色值。

流程图如下图 5-30 所示。

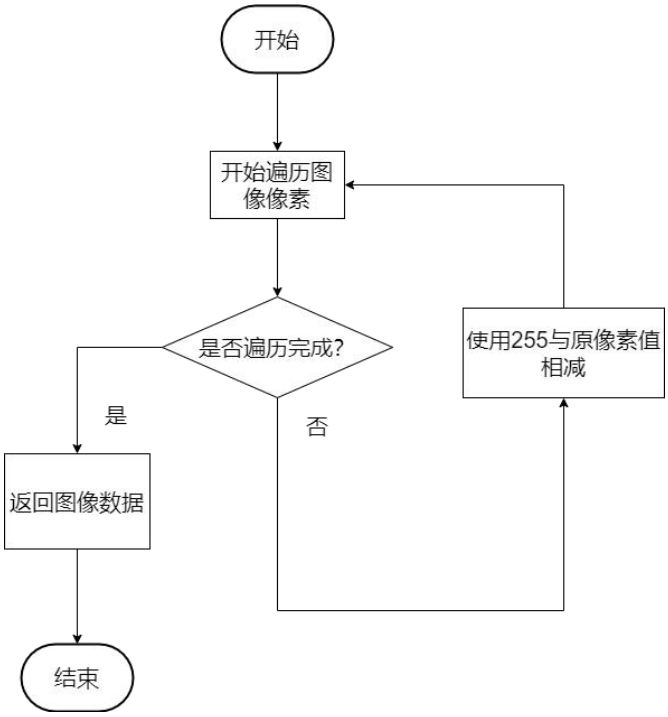

图 5-30 颜色反转流程图

5.2.14 补色

函数通过接收图像数据,之后通过迭代图像数据的每个像素的三个颜色通道,计算当前像素的最大和最小 RGB 值，然后进行补色计算。

流程图如下图 5-31 所示。

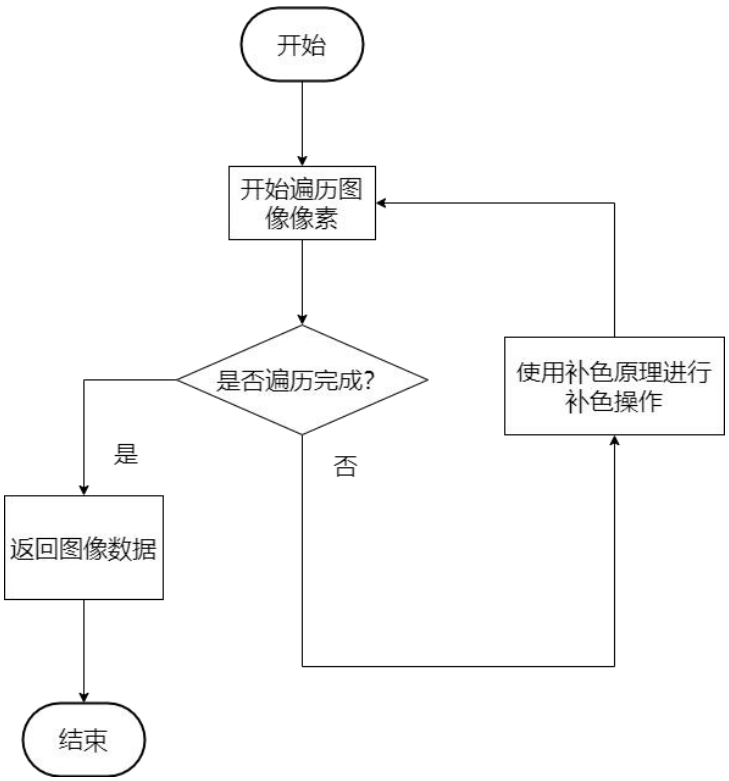

图 5-31 补色流程图

5.2.15 锐化

原图与高反差图像相加,可以得到锐化之后的图像[17]。

因此,需要先进行图像的高反差保留操作（高反差保留算法将会在5.2.20 中提及）。

而原图与高斯模糊图相减得到的图像就是高反差保留之后的图像。

首先需要进行高斯模糊操作，高斯模糊函数的解释在 5.2.4 介绍过，此处不再赘述。

接下来就是进行高反差保留操作。

根据原图和高斯模糊图的关系，得出高反差图像的获取方式：

原图减去高斯模糊图像得到高反差图像。

高反差函数接受两个参数，原始图像的数据和高斯模糊图像数据，

在循环中，根据原图减去高斯模糊图像得到高反差图像，可以计算出高反差之后的变量 between。

之后，对应位置的像素值之差加上一个 128，以调整到 0-255 的范围，得到高反差保留后的像素值（因为结果可能会出现负数或者一个很小的正数，因此需要加上 128）。

根据 uin8_t 数据类型：

如果结果超过 255，会将结果截断至 255；

如果得到的结果小于 0，会将结果截断至 0。

上述操作确保所有像素值都在 0 到 255 的范围内。

得到高反差保留的图像之后，就可以进行图像的锐化了：

图像锐化函数接受两个参数，原始图像数据和高反差图像数据。

在接下来的循环中，遍历每一个像素，然后将高反差图像的像素值加到原始图像上，这种方式会突出图像中的明亮和暗部差异，从而增强图像的锐度。

addValue－170：首先，计算原始像素值与高反差图像像素值之和减去 170(多次实验之后得到的数值)的结果。这个操作旨在增加图像的对比度和锐度。

std::min(addValue - 170, 255)：接着，使用 std::min 函数确保结果不会超过 255。这是因为图像像素值的范围通常是 0 到 255，超过这个范围会导致像素值不合法。

std::max(std::min(addValue - 170, 255), 0)：最后，再使用 std::max 函数确保结果不会低于 0。这是因为图像像素值不能为负数。

流程图如下图 5-31 所示。

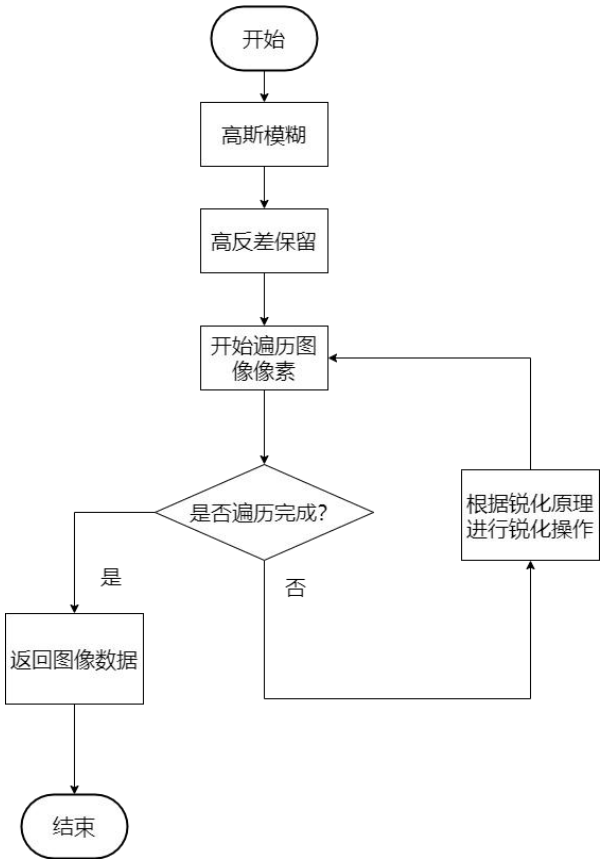

图 5-31 锐化流程图

5.2.16 色彩平衡

色彩平衡函数接受 3 个参数：原始图像数据，图像宽度，图像高度。

算法通过计算图像中红色、绿色和蓝色通道的平均值，并将每个通道的像素值乘以相应的缩放因子来调整图像的颜色平衡。

1.首先遍历图像的所有像素，累加红、绿、蓝通道的像素值；

2.其次通过除以图像的总像素数，得到每个通道的平均值；

3.接着计算每个通道的调整因子，这是每个平均值相对于 255 的比例；

```
double_t redFactor = avgRed > 0.0 ? avgRed / 255.0 : 1.0;
double_t greenFactor = avgGreen > 0.0 ? avgGreen / 255.0 : 1.0;
double_t blueFactor = avgBlue > 0.0 ? avgBlue / 255.0 : 1.0;
```

这样写的目的在于：

归一化因子：将颜色通道的平均值归一化到范围[0,1]之间，这样做是为了确保因子在颜色调整时不会导致颜色值超过合理范围（[0,255]）。

避免除零错误：如果某个通道的平均值为 0，那么直接除以该平均值可能导致除零错误。为了避免这种情况，使用了三元运算符 (avgX> 0.0？avgX / 255.0 : 1.0)，如果平均值大于 0，则使用归一化的平均值；否

则，使用默认值 1.0。这确保了不会出现除零错误。

调整因子作用:将颜色通道的每个像素值乘以对应的调整因子。这样，如果某个通道整体偏暗，乘以小于 1 的因子会增强该通道的亮度，达到平衡的效果。

4.然后遍历图像的所有像素,将每个通道的像素值乘以相应的调整因子;

5.最后将调整后的像素值更新到图像数据中。

这样就实现对图像颜色平衡的调整。

流程图如下图 5-32 所示。

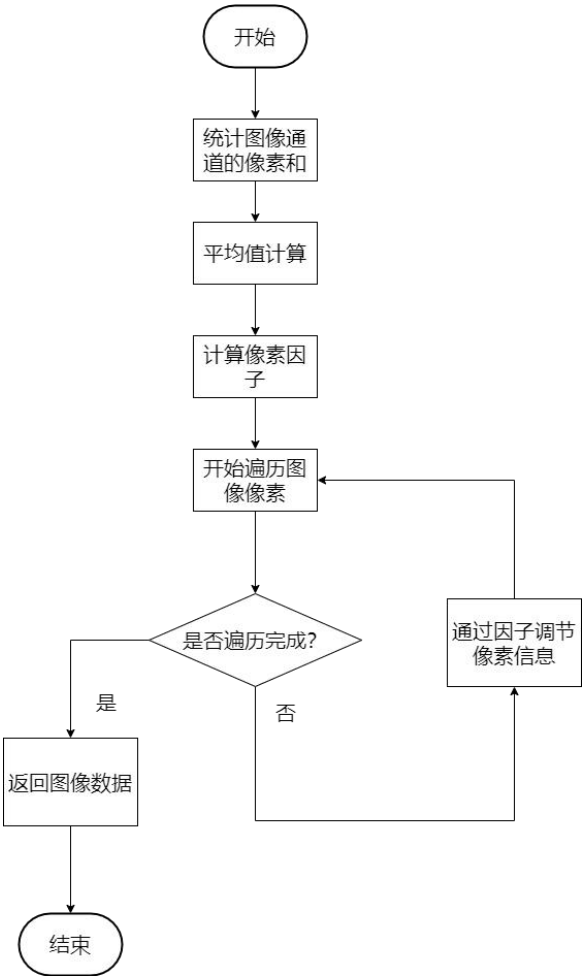

图 5-32 色彩平衡流程图

5.2.17 图像微调

该函数用于在图像上应用液化效果。

函数接受原始图像数据，图像宽度，图像高度，x 坐标(QT 鼠标点击事件获取)，y 坐标(QT 鼠标点击事件获取)，半径 radius(通过 QT 控件输入)，液化程度 intensity (通过 QT 控件输入)。

首先，函数需要遍历图像像素，计算每个像素到中心点的距离。

如果距离小于液化区域的半径，就会应用液化效果。液化效果的强度通过 intensity 参数控制，通过计算新的坐标位置，将像素点的位置进行扭曲。

首先需要计算像素点到中心点的距离。

勾股定理用于计算图像中每个像素点到中心点的距离,这是因为液化效果是根据距离来调整像素点的强度。

通过勾股定理计算距离,可以在平面坐标系中准确地得到两点之间的直线距离。

接着要计算扭曲的强度：

double warpAmount = intensity * std::sin(distance / radius * 3.14159265);

正弦函数用于计算液化效果的强度,它会根据距离和半径的比值来产生一种周期性的扭曲效果。

根据像素到中心点的距离，计算出变形的幅度。

intensity 是用户指定的调整强度,而 std::sin(distance / radius * 3.14159265) 则是一个用于在半径范围内变化的正弦函数。

这个正弦函数的值在半径边界上为 0，在中心点处为最大值，使得在这个范围内产生变形效果。

这样，通过对像素位置进行变形，实现了一种类似于液化的效果，使得图像在指定半径范围内的像素位置发生变化。

然后计算新的坐标，新的索引位置。

最后，通过像素值的替换，将当前像素的颜色值替换为目标像素的颜色值。

流程图如下图 5-33 所示。

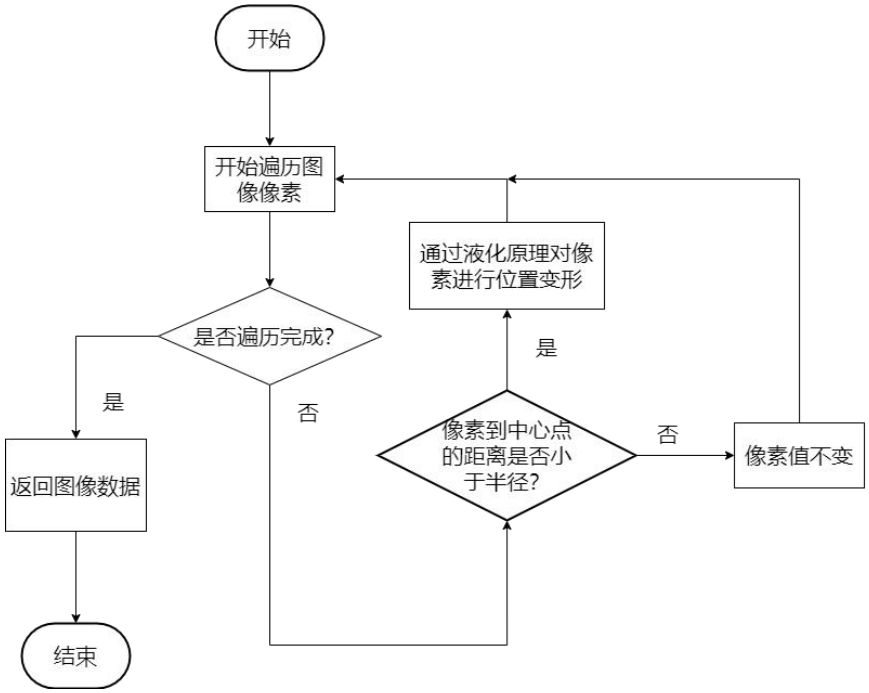

图 5-33 图像微调流程图

5.2.18 图像旋转

图像旋转就是将图像按一定角度旋转，依据当前点坐标计算出来的旋转后的坐标往不是整数，因此需要进行插值。常用的插值方法有最近邻插值法、线性插值法和样条插值法。

将旋转后图像的像素点映射回原图像，找到它的采样点，即旋转的逆变换。映射的结果不会都是整数像素点，那么旋转后的点的像素值由与采样点最邻近的像素值表示，这就是最近邻插值[20]。

如下图 5-34 所示:

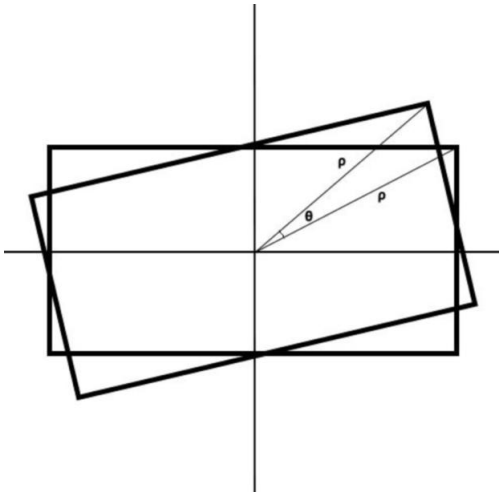

图 5-34 最近邻插值

该函数通过输入图像数据，图像宽度，高度，旋转角度(QT 控件输入)

首先，需要把角度转换为弧度，这是因为接下来使用的标准三角函数

库接受弧度作为参数，而不是角度。这个转换使用了常量 π，它是一个圆周角度的常数，等于 180 度。

需要使用以下公式将角度转换为弧度:

$$radians = angle * M\_PI/180.0 \quad (5\text{-}15)$$

这样的转换是为了确保在使用三角函数时得到正确的结果。

随后通过 centerX 和 centerY 计算图像的中心点。

随后经过一个循环，遍历原图像的所有像素点，

循环当中，执行旋转的代码使用了二维旋转矩阵的数学公式，对图像中的每个像素进行了顺时针旋转。

对于每个旋转后的坐标 (rotatedX, rotatedY)，通过最近邻插值法确定其在原图像中的对应坐标，并将像素值赋给旋转后的图像。

流程图如下图 5-35 所示。

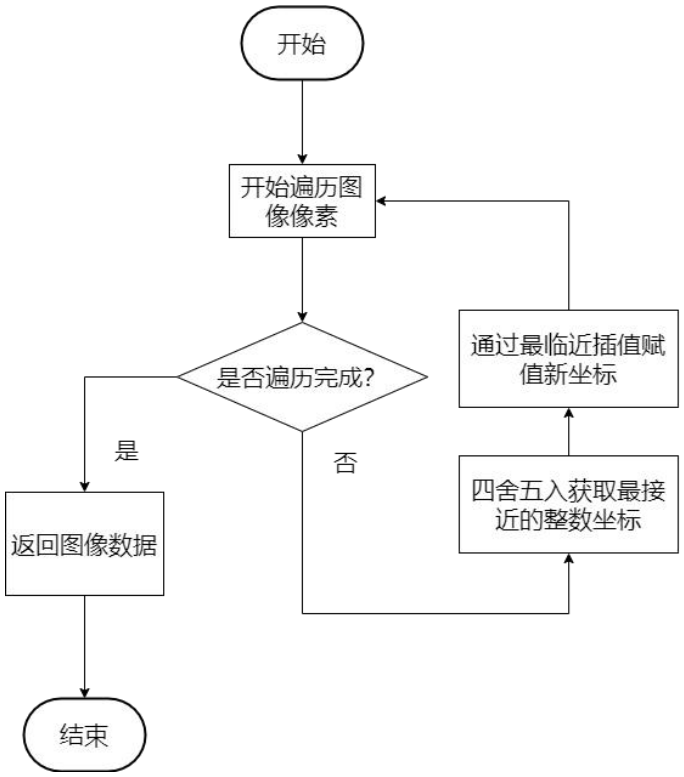

图 5-35 旋转图像流程图

5.2.19 阈值处理

图像的二值化或阈值化旨在提取图像中的目标物体,将背景以及噪声区分开来。通常会设定一个阈值 T，通过 T 将图像的像素划分为两类：大于 T 的像素群和小于 T 的像素群。

灰度转换处理后的图像中，每个像素都只有一个灰度值，其大小表示明暗程度。二值化处理可以将图像中的像素划分为两类颜色，常用的二值

化算法如下所示:

$$\begin{cases} Y = 0, gray < T \\ Y = 255, gray \geq T \end{cases} \tag{5-16}$$

当灰度 Gray 小于阈值 T 时，其像素设置为 0，表示黑色；当灰度 Gray 大于或等于阈值 T 时，其 Y 值为 255，表示白色[21]。

第一步,对于每个像素的颜色通道，通过加权求和计算灰度值。

第二步,将灰度值与预先定义的阈值进行比较。

第三步，计算一个比例因子 factor，这个比例因子的计算公式是:

$$gray - threshold\ /(255.0 - threshold) \tag{5-17}$$

使用比例因子调整每个颜色通道的值，增加一个固定的偏移量（这里是 100）。确保最终的颜色通道值不超过合理范围 0 到 255，具体做法如下：

计算一个比例因子 factor，其值在 0 到 1 之间，表示灰度值低于等于阈值的程度。这个比例因子的计算公式是

$$gray\ /\ threshold \tag{5-18}$$

使用比例因子调整每个颜色通道的值，减小颜色通道的值。同样，确保最终的颜色通道值不低于 0。

这个过程的目的是在图像中根据灰度值与阈值的关系,对颜色进行调整，以实现一种二值化的效果。

如果像素的灰度值高于阈值，则增加颜色通道值，使得图像中灰度高的区域更亮。如果灰度值低于等于阈值，则减小颜色通道值，使得图像中灰度低的区域更暗。

流程图如下图 5-36 所示。

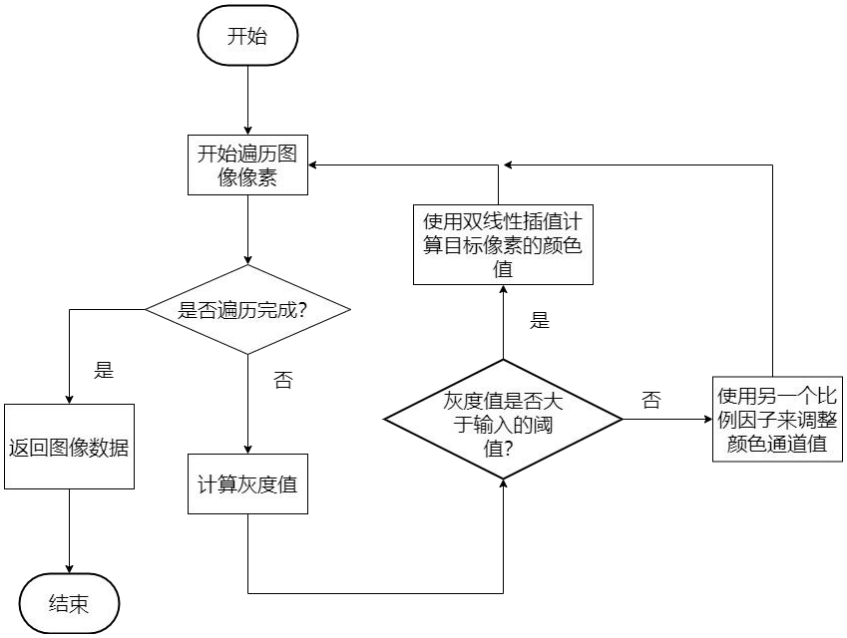

图 5-36 阈值处理流程图

5.2.20 高反差保留

计算原始图像与经过高斯模糊后的图像之间的差异,目的是为了突出图像中的边缘和细节。

原图与高斯模糊图做差值,就可以得到高反差保留之后的图像[16]。

首先需要进行高斯模糊，高斯模糊操作在之前小节中已经介绍过，此处不再赘述。

最后进行高反差保留操作，根据高反差图像的生成原理，只需要把原始图像数据与上一步得到的高斯模糊图像做差值即可。

流程图如下图 5-37 所示。

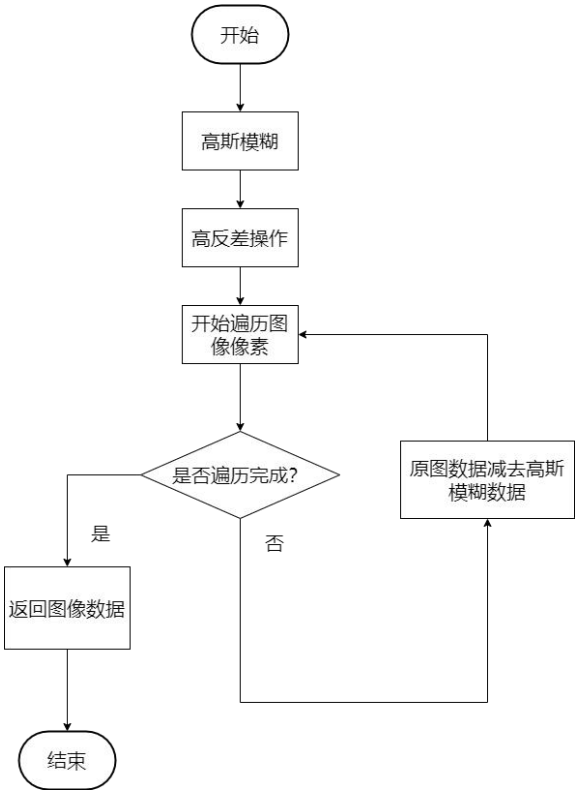

图 5-37 高反差保留流程图

5.2.21 高光

函数接受一个图像数据 imageData 和一个外部输入的像素值 pixel。

在指定像素值之上的像素颜色值会降低 50，最后确保降低后的颜色值不小于 0。这样可以模拟图像中亮度较高区域的效果。

流程图如下图 5-38 所示。

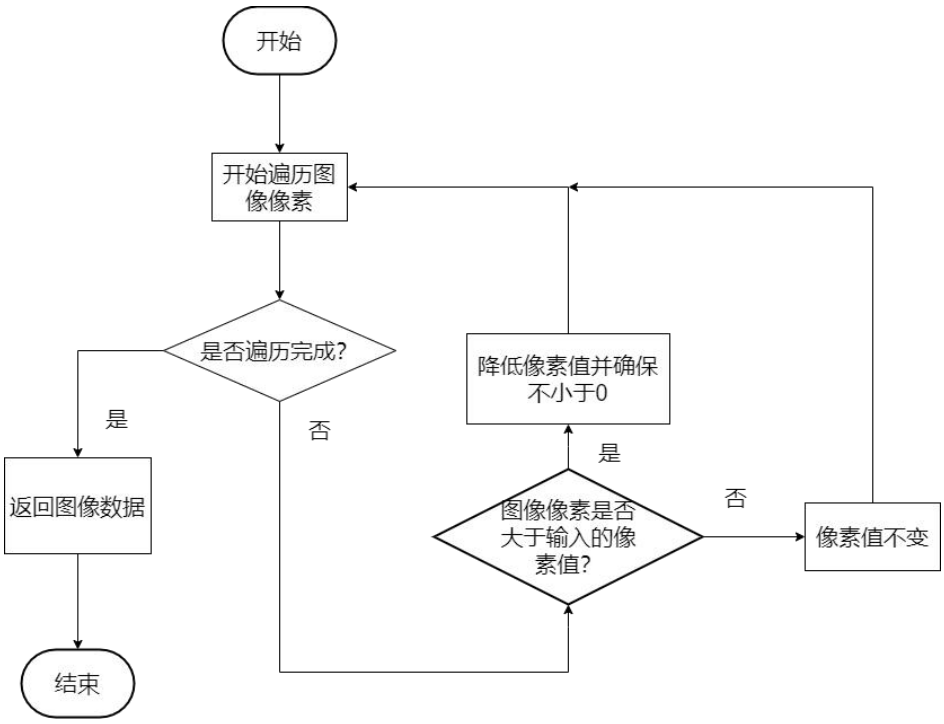

图 5-38 高光流程图

5.2.22 图像阴影化

函数接受一个图像数据 imageData 和一个阴影值 shadowValue。其中颜色通道值小于阴影值的像素会被设为 0，即形成阴影效果。流程图如下图 5-39 所示。

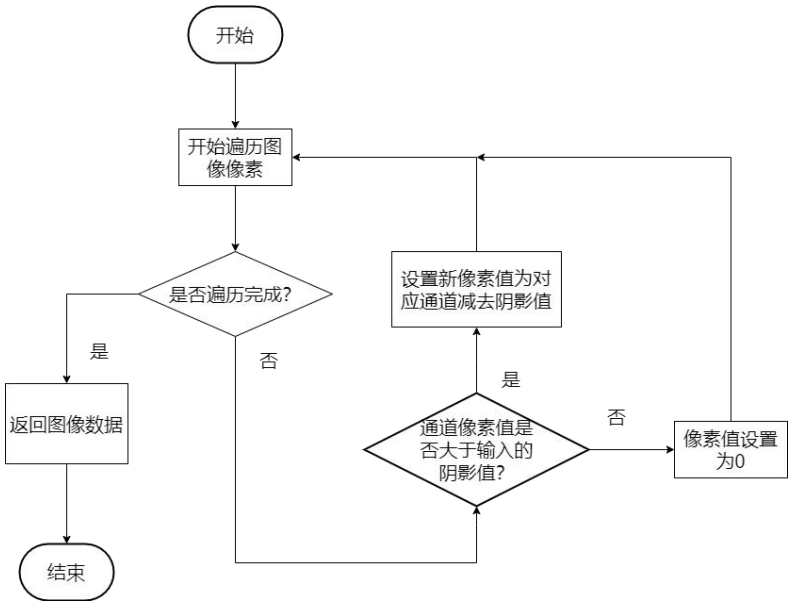

图 5-39 图像阴影化流程图

5.2.23 改变亮度、对比度、饱和度

Brightness 函数接受一个图像数据 brightnessImageData 和一个亮度值 brightnessValue。

对每个像素的 RGB 通道进行调整，增加或减少亮度。

亮度值的范围为-150 到 150，确保颜色通道的值在合理范围内，避免图像变得过曝或过暗，失去了细节信息，甚至无法正常观看。

流程图如下图 5-40 所示。

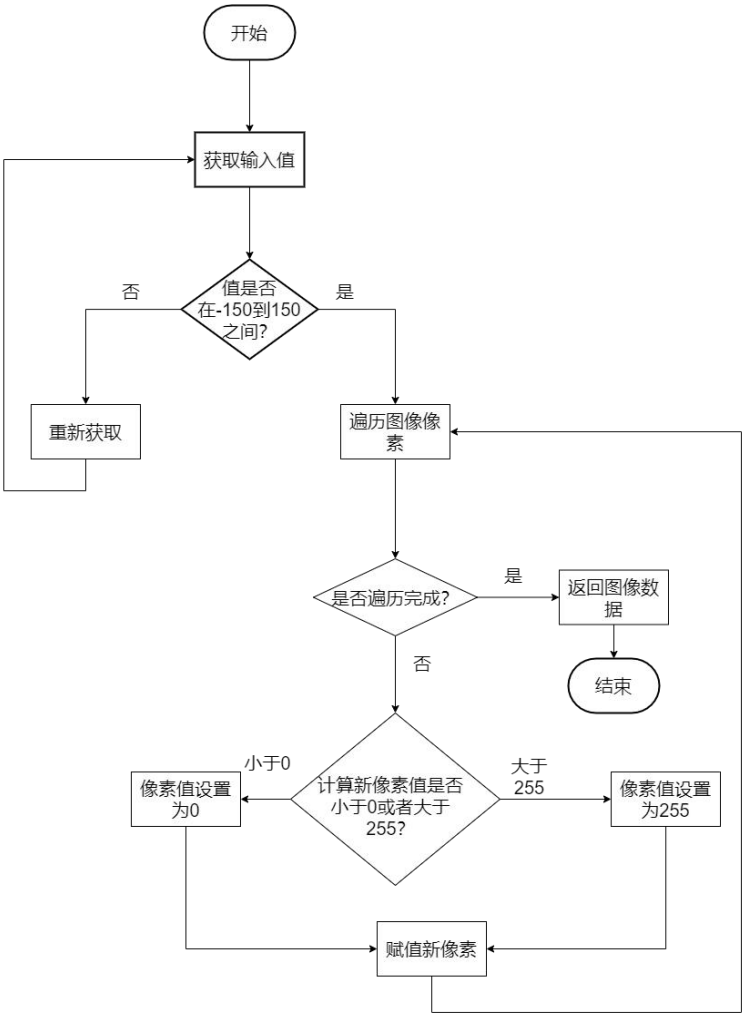

图 5-40 改变亮度流程图

Contrast 函数接受一个图像数据 contrastImageData 和一个对比度值 contrastValue。

对每个像素的 RGB 通道进行调整，增加或减少对比度。

对比度值的范围为-50 到 100，确保颜色通道的值在合理范围内，较低的对比度值会导致图像变得平淡，失去了色彩的鲜明度，而较高的对比度值可能会使图像变得过于强烈，出现失真现象。

流程图如下图 5-41 所示。

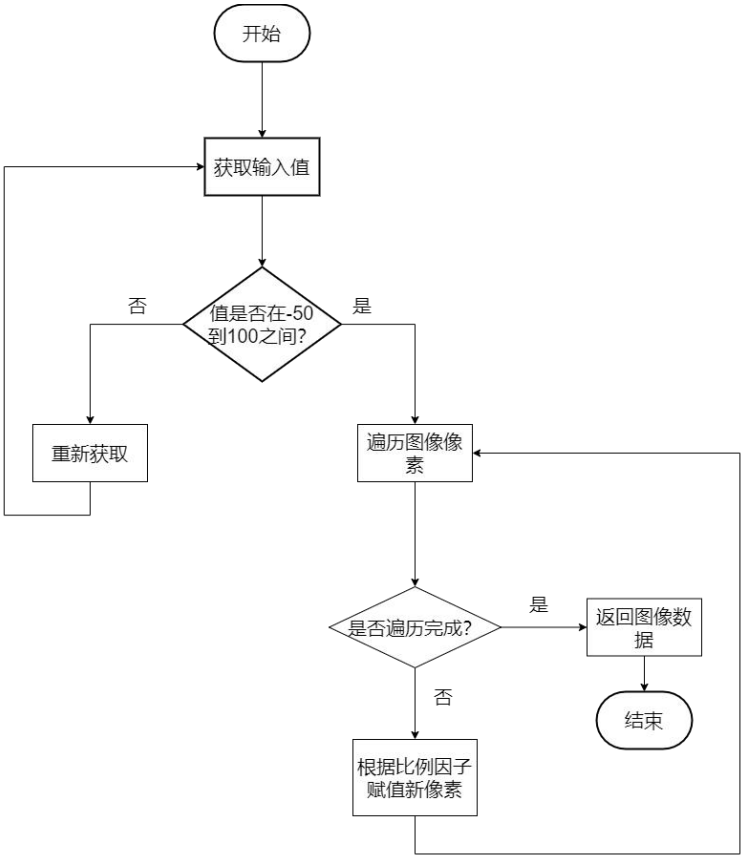

图 5-41 改变对比度流程图

饱和度调整：

饱和度调整需要先进行色彩空间的装换

首先介绍 RGB 色彩空间:

RGB 色彩空间使用 Red 红色、Green 绿色、Blue 蓝色三个分量来表示颜色，每个分量的范围是[0%, 100%][26]。

其次介绍 HSV 色彩空间:

HSV 色彩空间使用 Hue 色相、Saturation 饱和度、Value 明度来表示颜色[24]。

需要转换对比度，就需要进行 RGB<->HSV 色彩空间互相转换

RGB 转换 HSV 的公式如下:

$$
\begin{aligned}
&R' = R/255 \\
&G' = G/255 \\
&B' = B/255 \\
&C_{\max} = \max(R', G', B') \\
&C_{\min} = \min(R', G', B') \\
&\Delta = C_{\max} - C_{\min} \\
&HUE(H) \\
&H = \begin{cases} 0° & \Delta = 0 \\ 60° \times (\dfrac{\mathrm{G}' - B'}{\Delta} + 0) & C_{\max} = R' \\ 60° \times (\dfrac{\mathrm{B}' - R'}{\Delta} + 2) & C_{\max} = G' \\ 60° \times (\dfrac{\mathrm{R}' - G'}{\Delta} + 4) & C_{\max} = B' \end{cases} \\
&Saturation(S) \\
&S = \begin{cases} 0 & C_{\max} = 0 \\ \dfrac{\Delta}{C_{\max}} & C_{\max} \neq 0 \end{cases} \\
&Value(V) \\
&V = C_{\max}
\end{aligned}
\tag{5-19}
$$

最小和最大值：

通过计算 r、g 和 b 中的最小和最大值，即 minVal 和 maxVal，确定颜色值的范围。

Delta 计算：

delta 被计算为 maxVal 与 minVal 之间的差异。这是对 RGB 空间中颜色分布或范围的度量。

Value 计算：

v（明度）被设置为最大值（maxVal）。在 HSV 模型中，这表示颜色的亮度或强度。

Saturation 计算：

根据 maxVal 是否为零，计算 s（饱和度）。如果 maxVal 为零，饱和度设置为 0。否则，它被计算为 delta 除以 maxVal。饱和度表示颜色的纯度或鲜艳度。

Hue 计算：

h（色相）的计算基于最大值与其他颜色分量之间的关系。公式取决于 r、g 或 b 哪一个是最大值。结果乘以 60 将色相转换为度数。如果色

相为负数，添加 360 以确保它在有效范围内（0 到 360 度）。

接下来需要再次转换为 RGB,公式如下:

$$
\begin{aligned}
h_i &= floor(\frac{H}{60}) \\
f &= \frac{H}{60} - h_i \\
p &= V*(1-S) \\
q &= V*(1-f*S) \\
t &= V*(1-(1-f)*S) \\
(R,G,B) &= \begin{cases} (V,t,p) & h_i = 0 \\ (q,V,p) & h_i = 1 \\ (p,V,t) & h_i = 2 \\ (p,q,V) & h_i = 3 \\ (t,p,V) & h_i = 4 \\ (V,p,\mathrm{q}) & h_i = 5 \end{cases}
\end{aligned}
\tag{5-20}
$$

根据公式,转为对应的代码。

饱和度为 0：

如果饱和度 s 为零，表示颜色是灰度色，所有的 RGB 分量都应该具有相同的值。因此，r、g、b 都被设置为 v（明度）。

非灰度色情况：

如果颜色是彩色的（饱和度不为零），则使用一系列计算来将 HSV 转换为 RGB。

h 被归一化到区间 [0, 6) 以适应 switch 语句中的 6 个情况。这是因为色相的度数是 360 度，而 switch 语句中有 6 个情况，因此每个情况对应 60 度。

颜色分量计算：

根据 i 的值，计算 r、g 和 b 的值。这涉及到线性插值，使用了三个参数 p、q 和 t。这些参数的计算涉及到 HSV 颜色模型的特定公式，确保了正确的颜色转换。

流程图如下图 5-42 所示。

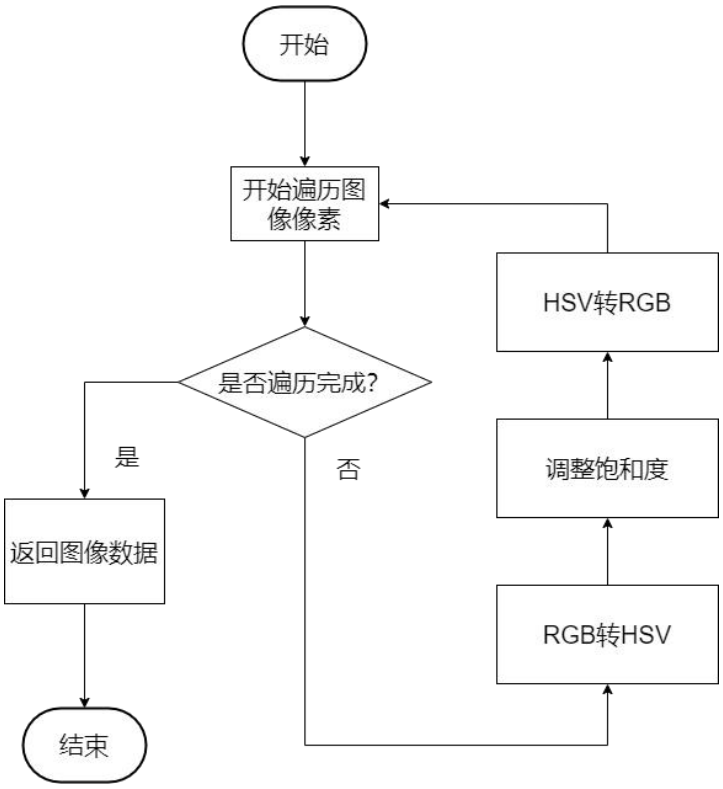

图 5-42 饱和度调整流程图

第 6 章　系统其他功能介绍与系统性能优化

6.1 系统其他功能介绍

6.1.1 加载图像

加载图像流程图如下图 6-1 所示。

用户进行打开文件操作，首先会判断是否为 BMP 文件，如果不是，则报错，用户需要重新选择；如果是 BMP 文件，则把图片信息写入内存中。

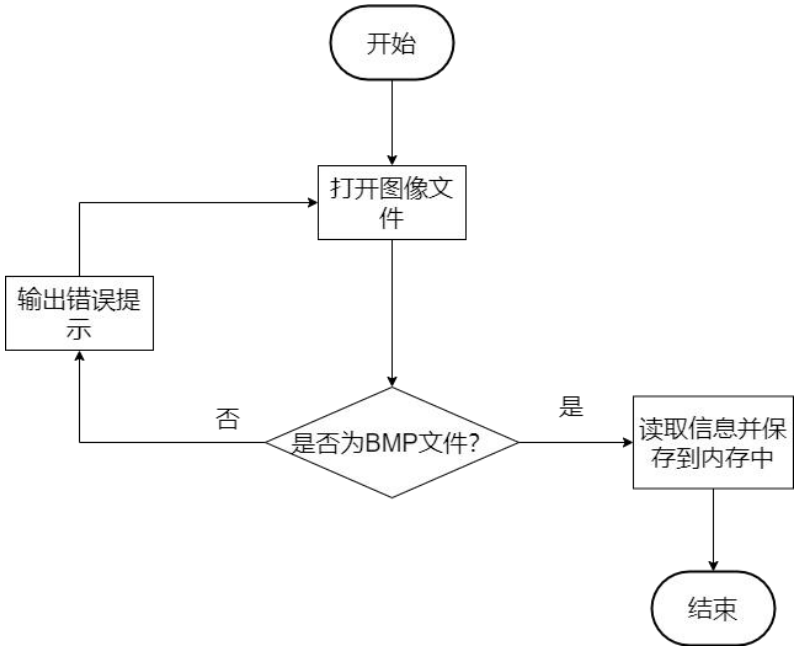

图 6-1 加载图像流程图

在 UI 界面中,点击“打开图片”按钮或者双击界面,进入打开图像文件槽函数。

首先设置文件的默认位置，为 homePath，即用户主目录。

因此点击“打开图片”按钮或者双击界面，会进入用户主目录，用户可以选取图片文件。

接下来设置一个过滤器，表示用户只可以选择后缀名为.bmp 结尾的文件。

进入判断，如果选择的目录为空（用户关闭了选择文件的窗口），则不进行任何操作，退出函数。

如果不为空，则显示所有隐藏的控件，并且调用加载 BMP 文件的函数 ReadBMPFile。

若一切正常,显示图片到对应的控件 imageLabel 上，并且 canSave 变量设置为 true，表示图片在后续可以进行保存。

用户便可以执行下一步操作。

6.1.2 保存图像

保存图像流程图如下图 6-2 所示。

用户点击保存图像按钮，程序会判断是否可以进行保存，并且图像历史链表是否存在相应的图像数据，若两者都满足，则进行保存；若不满足，则输出保存失败提示。

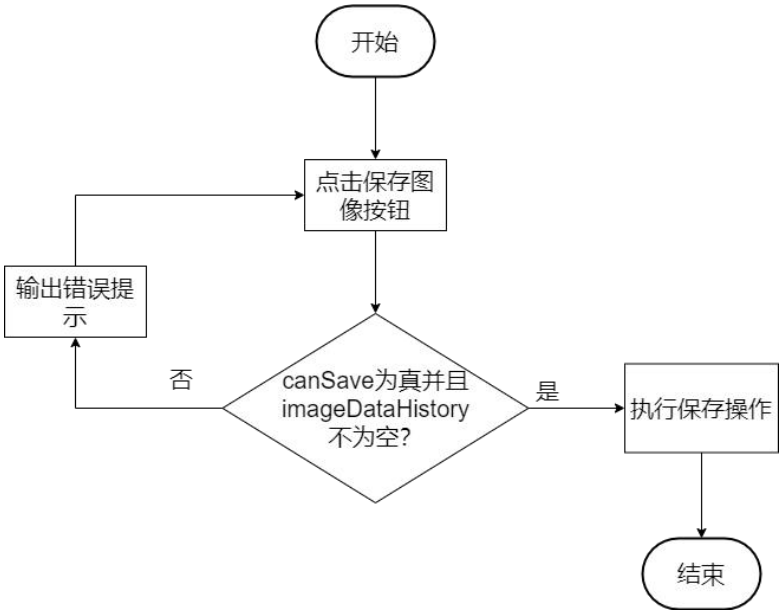

图 6-2 保存图像流程图

创建一个临时变量 saveImageData,用于存储图像信息。

如果 canSave 为真并且 imageDataHistory 不为空（imageDataHistory 是一个链表，后续会提到），则进行图片保存。

若条件不成立，则弹出对话框，提示用户。

6.1.3 撤销和重做操作

撤销和重做操作需要用到一个数据结构—链表。

链表可以存储之前的图像信息。

在每个处理函数中，保存当前的图像数据到链表中，然后在调用撤销或者重置功能时,从链表中获取之前或者之后的图像数据以此进行回退或者恢复。

首先创建链表对象，c++自带的 list 容器简化了链表的操作，因此首先需要创建 list 对象：imageDataHistory 和 redoImageDataHistory

这两个链表的数据类型为 uint8_t,存储了图像信息,方便进行撤销和重做操作。

接着定义一个函数 SaveImageDataToHistory,用来保存当前图像数据到链表。

程序需要先判断数据链表是否为空，为空则存储数据；若不为空，则

需要判断当前图像数据是否等于链表最后一个的图像数据，如果不是，则插入图像数据，如果是，则不进行任何操作。

流程图如下图 6-3 所示。

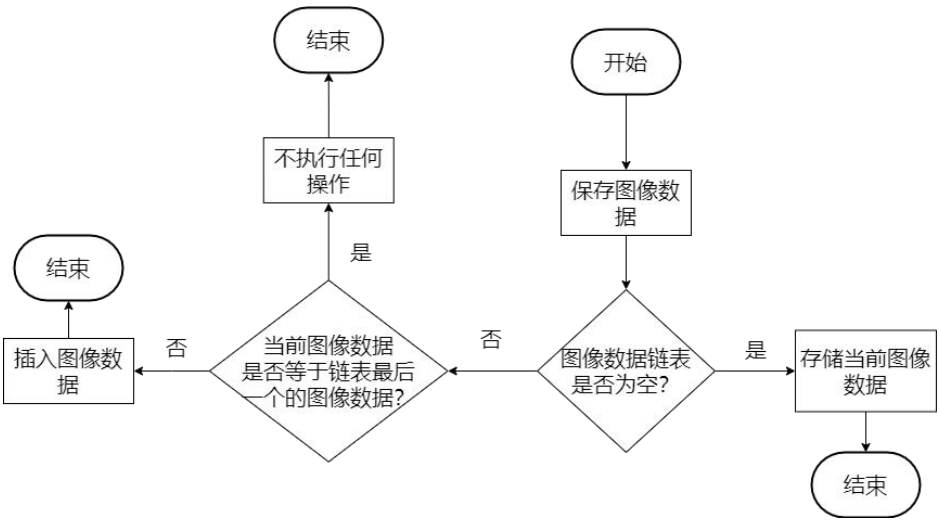

图 6-3 保存图像数据流程图

然后需要在每一个功能函数调用这个函数(以灰度图为例):

执行灰度图操作之后，保存灰度图图像数据到链表中。至此，链表当中就有了灰度图图像的数据信息。便于实现撤销和重做操作。

如果想要退回上一步操作,则需要调用函数 UndoImageProcessing。

用户点击撤销按钮,程序需要先判断链表是否为空,若为空,则 Debug 输出图像历史为空；若不为空，则保存当前图像数据到重做链表，然后判断数据链表是否为空，若为空则显示初始图像；若不为空则恢复图像数据并且更新图像显示。

流程图如下图 6-4 所示。

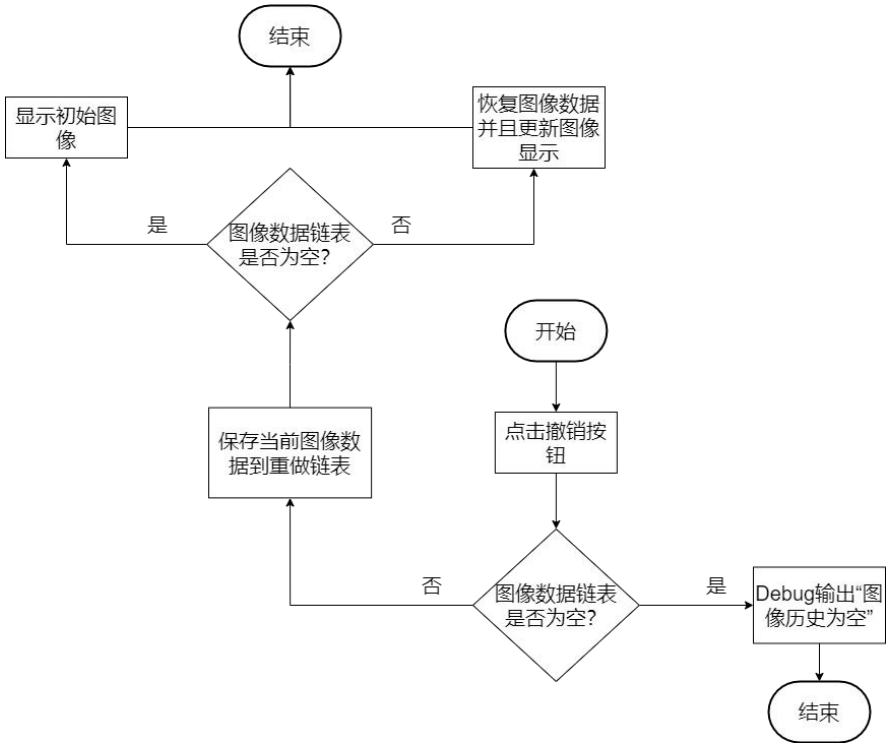

图 6-4 撤销功能流程图

此函数可以回退链表节点,实现撤销功能。

如果要进行重做，则调用 RedoImageProcessing。

用户点击重做按钮，程序判断重做图像数据链表是否为空，若为空，则 Debug 输出“重做图像链表数据为空”；若链表不为空，则恢复图像数据并且更新图像显示。

流程图如下图 6-5 所示。

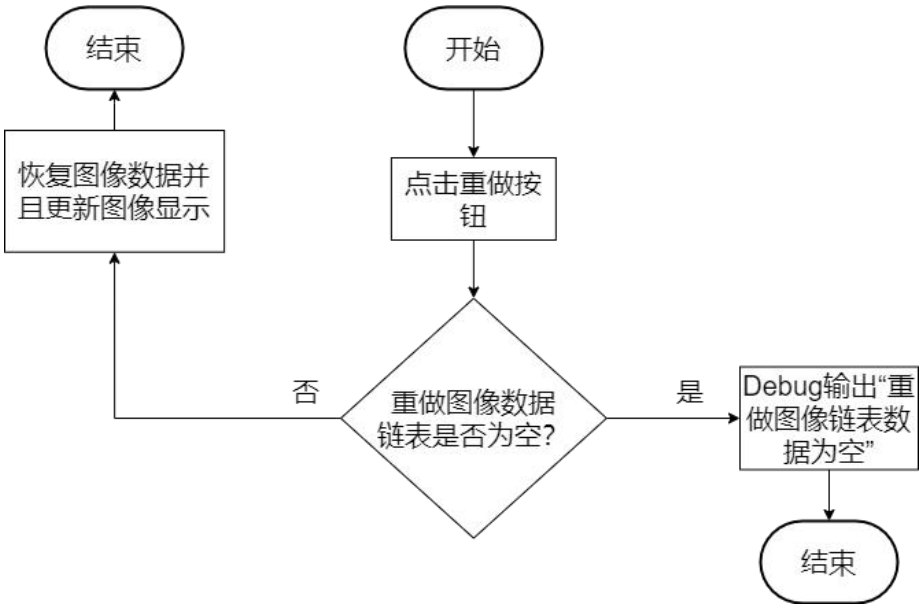

图 6-5 重做功能流程图

6.2 性能优化

C++11 的最大特点之一是引入了对多线程编程的支持。通过引入新的标准库头文件 <thread>、<mutex> 等，C++11 为开发人员提供了一套丰富的工具和接口，使得多线程编程变得更加简单和高效。使用这些工具，开发人员可以轻松创建线程、管理线程之间的同步与互斥，以及实现线程间的通信。

这一特性使得 C++成为了更加强大和灵活的编程语言，能够更好地满足现代软件开发中对于并发性和性能的要求[30,31]。

因此，利用 C++11 提供的多线程库，就可以实现大部分的性能优化。

首先，在程序开始运行时，需要根据用户设备的 cpu 线程数判断程序需要设置的线程。

随后使用多线程对系统进行优化，可以发挥出用户系统的最佳性能，加快图像处理的效率。

在多线程优化中，使用了一个函数 std::bind();

std::bind 函数是 C++标准库中的一个工具，用于创建可调用对象。它可以将成员函数、普通函数、函数对象或者 Lambda 表达式绑定到特定的参数，并返回一个可调用对象。这个可调用对象可以像函数一样调用，并且在调用时会自动传递绑定的参数[31]。

std::bind 函数的参数包括：

1.函数指针或可调用对象：指定要绑定的函数或者可调用对象。

2.指向成员函数的指针：如果要绑定的函数是类的成员函数，还需要提供类的实例指针。

3.函数参数占位符：使用 std::placeholders::_1, std::placeholders::_2, std::placeholders::_3……来表示占位,表示在调用时将被实际参数替换。

4.预绑定参数:在创建可调用对象时,可以传递一些参数进行预绑定。

在接下来的循环中，程序使用了分段处理的方法，把图像数据分为指定数量的段，然后执行图像相关操作，所有段处理完成（join）之后，则进行数据的整合显示。

第 7 章　系统测试

7.1 系统测试介绍

系统测试，是将需测试的软件，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素及环境结合在一起测试。在实际运行（使用）环境下，对计算机系统进行一系列的组装测试和确认测试。系统测试的目的在于通过与系统的软件需求作比较，发现软件与系统定义不符合或与之矛盾的地方。[32]

7.2 测试目的

此系统进行功能测试的目的是为了验证数字图像处理系统在各种条件下的功能，以确保其达到预期要求，保障用户的使用体验和系统的可靠性，避免出现功能运行过程中出现的与预期不符的现象。例如：打开图片功能出现异常，撤销与回退操作对图像无反应等。

7.3 系统关键功能测试

打开图片测试，如下图 7-1 所示：

图 7-1 UI 界面

双击界面之后，会出现图片选择窗口，如下图 7-2 所示：

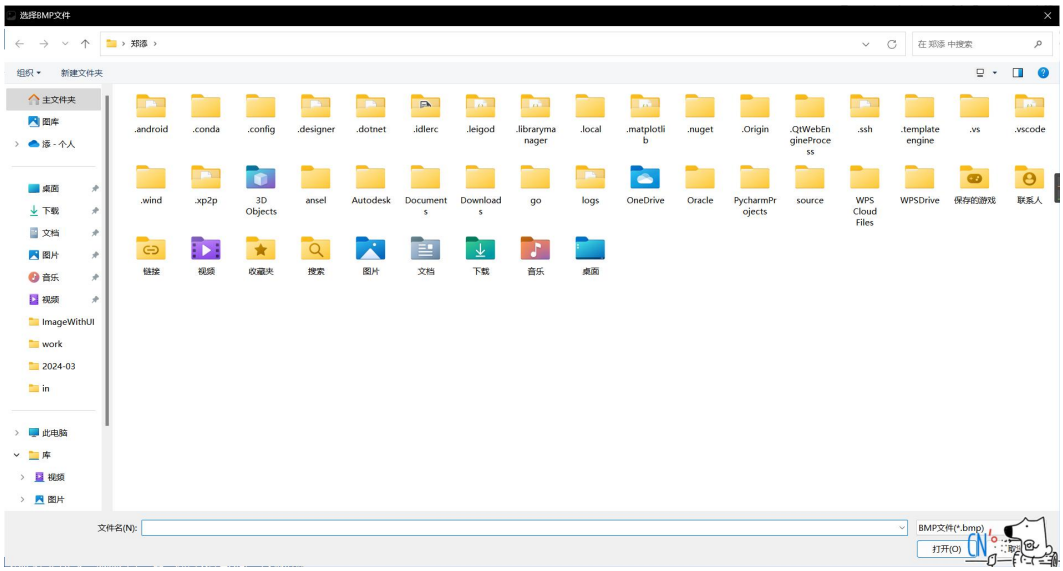

图 7-2 选择图片窗口

选择图片之后，图片就加载到主界面中，如下图 7-3 所示：

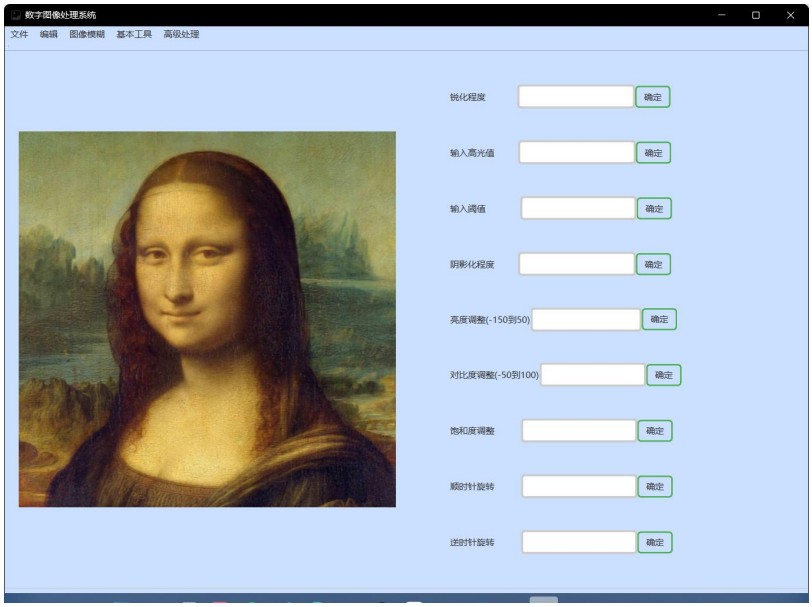

图 7-3 图像加载到主界面

BMP 图片可以正常打开。

保存图像如下图 7-4 所示：

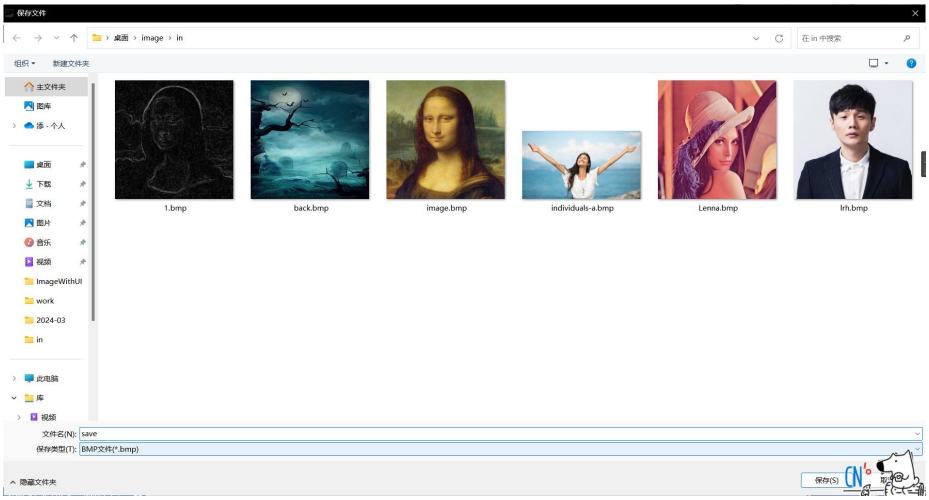

图 7-4 保存图像窗口

点击“保存图片”按钮，出现保存图片窗口，选择路径之后，输入文件名“save”然后点击保存。

如下图 7-5 所示。

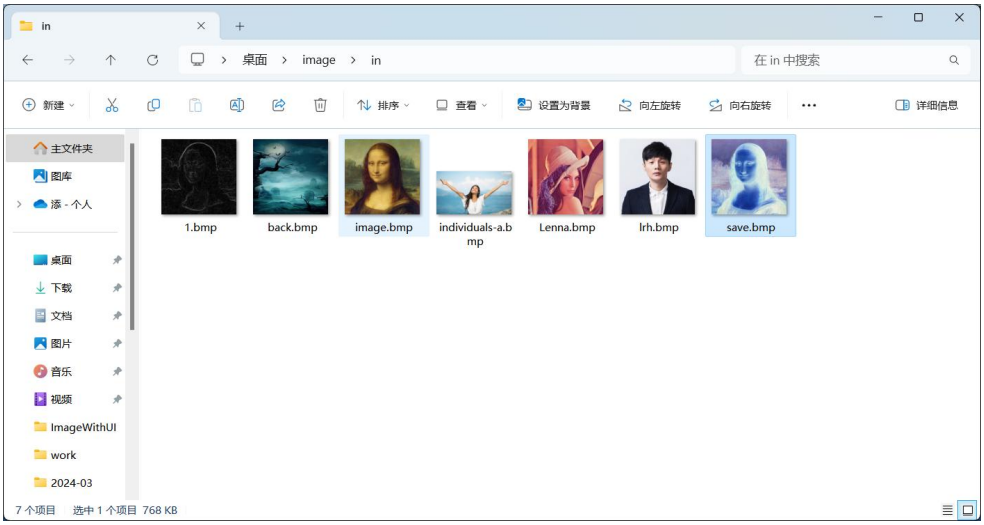

图 7-5 图片被成功保存到选择的路径下

打开选择的路径，发现图片被正常保存，如上图所示。

重置图像：

首先对图像进行多次处理，如下图 7-6 所示：

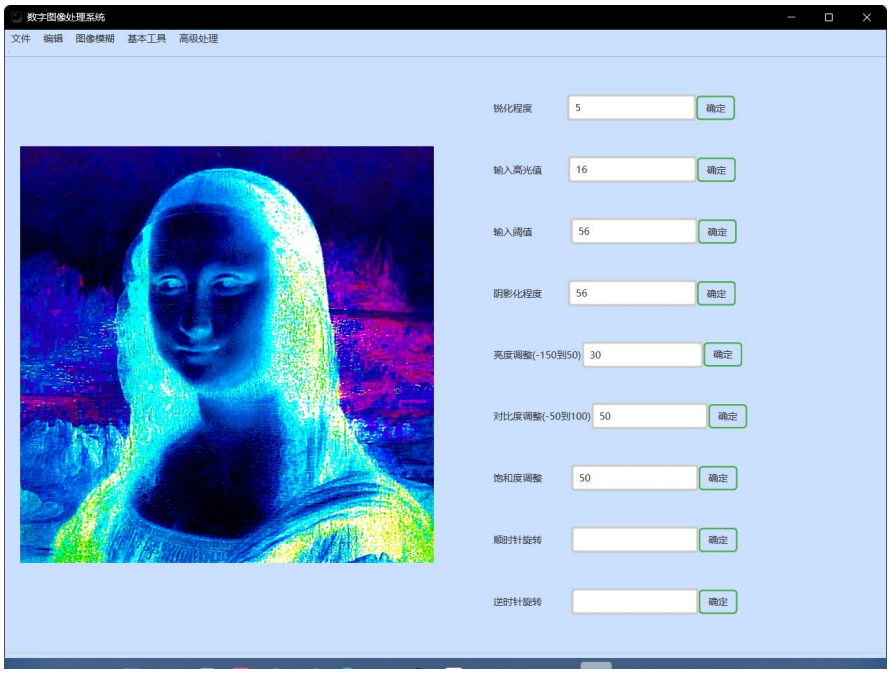

图 7-6 被处理过多次的图像

然后点击“编辑-重置图像”按钮，或者按下快捷键 ctrl+r，即可进行图像重置。如下图 7-7 所示：

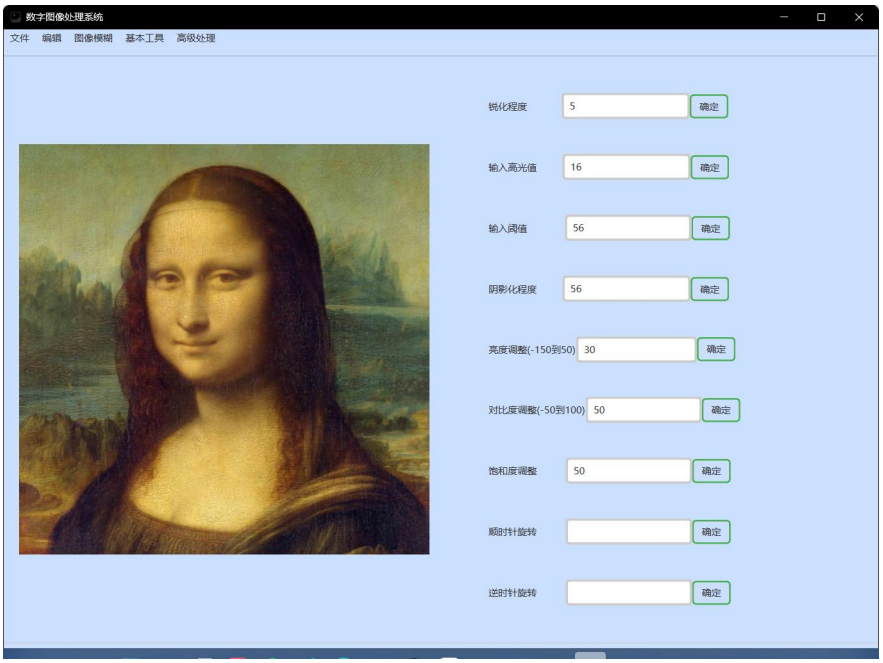

图 7-7 重置之后的图像

图像重置成功！

图像撤销与恢复：

图像先后进行“补色”->“边缘检测”->“亮度调整 50”。最后的结果如下图 7-8 所示：

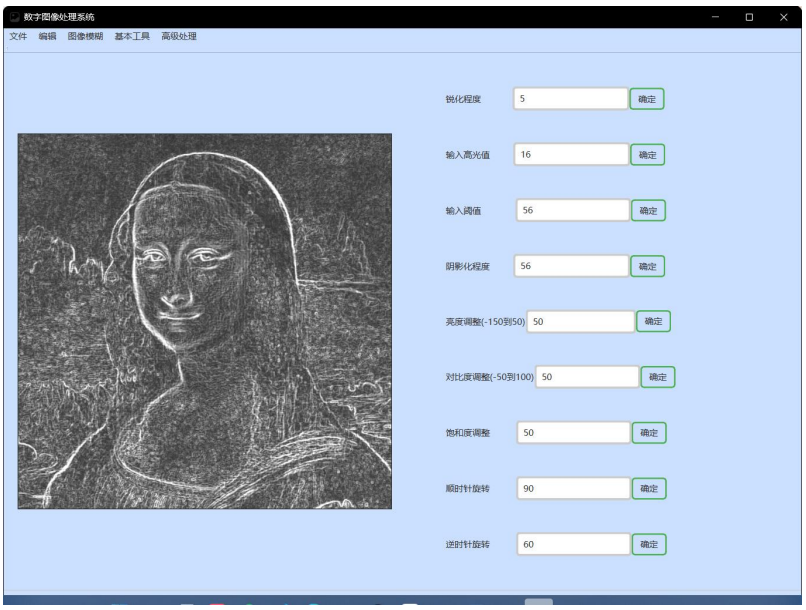

图 7-8 图像进行上述操作之后的结果

按下“编辑->撤销”按钮，或者按下快捷键 ctrl+z,可以实现撤销操作。按下两次撤销按钮，图片回到“补色”操作。

如下图 7-9 所示：

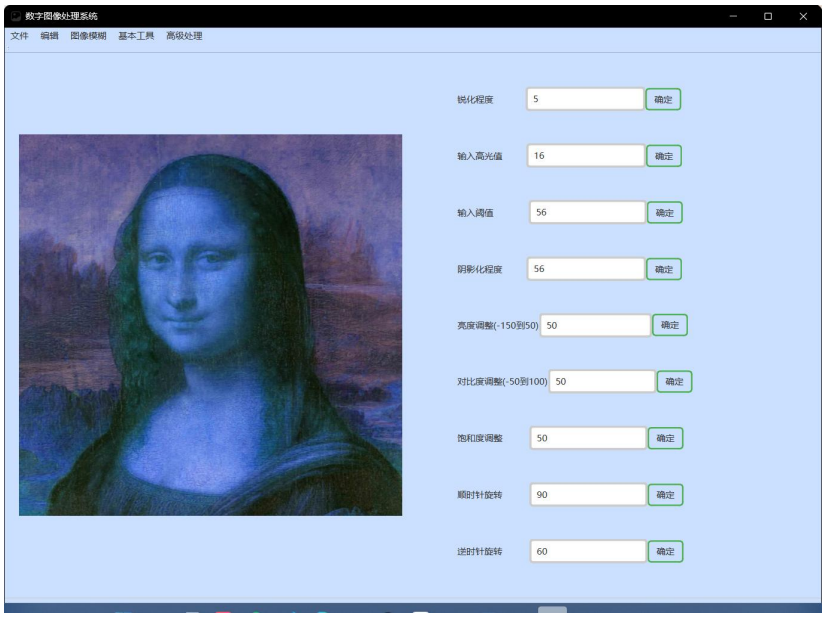

图 7-9 撤销两次的结果

接着按下“编辑->重做”按钮，或者按下快捷键 ctrl+z，可以实现重做操作。按下两次重做按钮，图片再一次回到图 7-8 的结果。

如下图 7-10 所示：

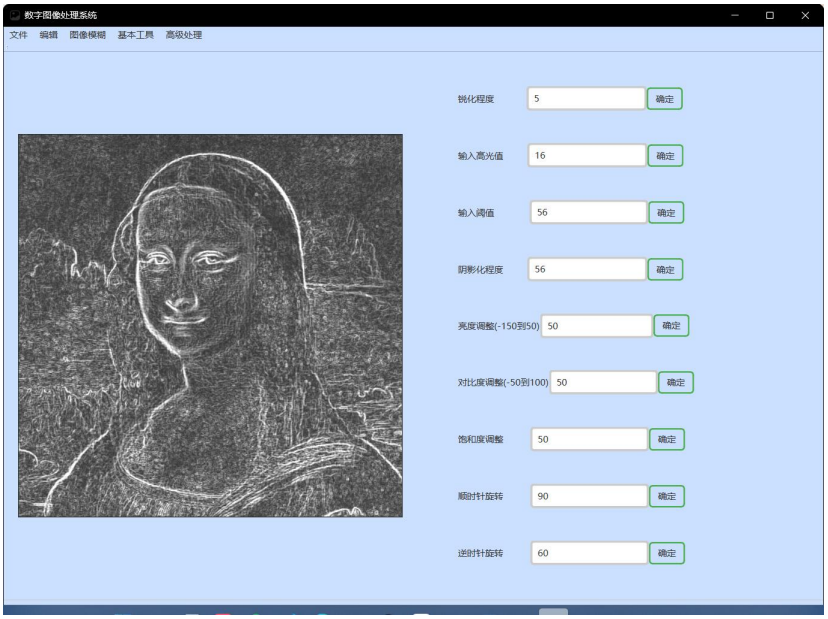

图 7-10 重做两次的结果

撤销和重做功能测试正常！

结论

本文设计的基于 C++和 QT 的数字图像处理系统为图像处理领域提供了一套强大而灵活的工具。通过原始的 C++代码实现各项关键功能，并借助 QT 前端框架提供用户友好的界面设计，本人成功实现了图像的读取和保存、灰度转换、自动对比度调整、高斯模糊、中值模糊等多个功能。系统的研发旨在为用户提供一套全面而实用的数字图像处理工具，以应对不同应用场景下的需求。

在系统的设计中，本人充分考虑了图像处理的基本操作，并通过对各项功能的深入研究和实现，确保系统在性能和功能上均能胜任各类图像处理任务。用户可以方便地使用这一系统，处理不同类型和尺寸的图像，达到所需的效果。同时，系统通过 C++的原始代码实现，摆脱了对第三方图像处理库的依赖，使得系统更加轻便高效。

在未来的研究中，可以考虑进一步扩展系统功能，引入更多先进的图像处理算法和技术，以适应不断发展的数字图像处理需求。通过持续的优化和更新，本系统有望成为数字图像处理领域的有力工具，为科研和工程应用提供更全面的支持。

致谢语

时光匆匆，转瞬间四年大学生活即将画上句点，回首这段时光，心中充满了感慨和感恩。在毕业设计的征程中，本人要由衷地感谢本人的毕业设计指导老师，黄彪老师。从选题到系统设计再到论文的撰写，老师在整个过程中都给予本人无私的指导和悉心的关照。无论是需求确定、功能实现还是论文写作，都得到了老师宝贵的建议和帮助，让本人受益匪浅。

感谢在大学生涯中遇到的所有老师们，是您们的辛勤耕耘和教导，让本人不断成长，收获了丰富的知识。每一位老师都是本人学业路上的引路人，对本人产生了深远的影响。

同时，本人要衷心感谢那些在毕业设计中伸出援手的同学和朋友们。在解决问题的过程中，大家共同努力、相互支持，让困难变得可以战胜。你们的友情和帮助是本人宝贵的财富。

最后，特别感谢各位老师在百忙之中抽出宝贵时间对本人的论文进行审阅。在您们的悉心指导下，本人的毕业设计得以圆满完成。再次感谢您们的辛勤付出和教诲！

感恩有您们的陪伴和支持，让本人的大学生活充满了温馨和成就。衷心感谢！

参考文献

[1] Lovelessing. BMP 图像文件完全解析[J/OL].知乎,2020-09-30.https://zhuanlan.zhihu.com/p/260702527.

[2] 量子孤岛.RGB 转换成灰度图像的一个常用公式 Gray = R*0.299 + G*0.587 + B*0.114 [J/OL].CSDN,2019-01-13.https://zhuanlan.zhihu.com/p/260702527.

[3] 深夜 i.C++自动对比度算法：简单易用的图像处理技术[J/OL].21xrx.com,2023-07-13.https://www.21xrx.com/Articles/read_article/232757.

[4]Vici__.真正搞懂均值模糊、中值模糊、高斯模糊、双边模糊[J/OL].CSDN,2019-10-10.https://blog.csdn.net/Vici__/article/details/102476784.

[5]jsxyhelu2015.基于 OpenCV 实现 Photoshop 的 17 种图层混合[J/OL].CSDN,2020-05-22.https://blog.csdn.net/jsxyhelu2015/article/details/108251476.

[6]Anita-ff.OpenCV 中图像伪彩色处理(C++/Python)[J/OL].博客园,2018-05-23.https://www.cnblogs.com/Anita9002/p/9076112.html.

[7]coder_Alger.OpenCV 利用高斯模糊实现简单的磨皮美颜效果[J/OL].CSDN,2022-04-26.https://blog.csdn.net/weixin_42289227/article/details/124415784.

[8]ishihara.最近邻插值、双线性插值与双三次插值[J/OL].知乎,2021-11-02.https://zhuanlan.zhihu.com/p/428523385.

[9]网络整理.sobel 算子如何实现水平边缘和垂直边缘的检测[J/OL].网易伏羲,2023-06-13.https://fuxi.163.com/database/879.

[10]Wikipedia.Sobel operator [J/OL].Wikipedia,2022-11-05.https://en.wikipedia.org/wiki/Sobel_operator.

[11]萌新掉包员.人脸形变算法—液化变形[J/OL].CSDN,2020-05-11.https://blog.csdn.net/tuntunmmd/article/details/106049752.

[12]OpenCV.Fisheye camera model [J/OL].OpenCV,2016-12-2.

[13] 小小的厂里挖呀挖.鱼眼相机标定方法[J/OL].知乎,2022-08-12.https://zhuanlan.zhihu.com/p/552930326.

[14] 工头阿乐.鱼眼相机成像模型以及基于 OpenCV 标定鱼眼镜头（C++）[J/OL].CSDN,2023-06-20.https://zhuanlan.zhihu.com/p/552930326.

[15] 一杯清酒邀明月.OpenCV 实现 Photoshop 算法(九): 高反差保留 [J/OL].博客园,2020-10-12.https://www.cnblogs.com/ybqjymy/p/13801395.html.

[16] 未雨愁眸.PS 滤镜算法原理——高反差保留 （High Pass）[J/OL].博客园,2014-06-09.https://www.cnblogs.com/mtcnn/p/9412710.html.

[17]稀土掘金.opencv 锐化 c++ [J/OL].稀土掘金,2020-07.https://juejin.cn/s/opencv%20%E9%94%90%E5%8C%96%20c%2B%2B.

[18]影叶.详谈色彩平衡原理和用法 J/OL].知乎,2017-06-08.https://zhuanlan.zhihu.com/p/27313620.

[19]牛肉好.数字图像处理-美图秀秀:大眼算法[J/OL].CSDN,2022-11-01.https://blog.csdn.net/Mibbp/article/details/127638925.

[20] 乙酸氧铍. [机器视觉学习笔记]最近邻插值实现图片任意角度旋转（C++）[J/OL].CSDN,2021-09-24.https://blog.csdn.net/weixin_44457994/article/details/120445830.

[21] eastmount.[Python 图像处理]七.图像阈值化处理及算法对比[J/OL].华为云,2021-08-11.https://bbs.huaweicloud.com/blogs/293565.

[22]阿里云. OpenCV-图像阴影调整[J/OL].阿里云,2023-10-18.https://developer.aliyun.com/article/1352885.

[23] OpenCV. Changing the contrast and brightness of an image![J/OL].OpenCV,2020-07-05.https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html.

[24]KeyeSssss.c++OpenCV 图像亮度的计算方法代码[J/OL].CSDN,2022-01-12.https://blog.csdn.net/weixin_43829212/article/details/122448272.

[25] broad-sky.OpenCV C++ 图像对比度和亮度[J/OL].CSDN,2021-10-09.https://blog.csdn.net/qq_37164776/article/details/120671053.

[26] 喵呜喵呜呜.RGB 与 HSV 色彩模型详解[J/OL].知乎,2023-05-03.https://zhuanlan.zhihu.com/p/626445292.

[27]全栈程序员站长.颜色空间 RGB 与 HSV(HSL)的转换[J/OL].腾讯云,2021-12-17.https://cloud.tencent.com/developer/article/1921044.

[28]LEE 的 FPGA.由 RGB 到 HSV 的转换详解[J/OL].知乎,2020-02-09.https://zhuanlan.zhihu.com/p/105886300.

[29]萌萌哒程序猴.数字图像处理——RGB 与 HSV 图像互相转换原理[J/OL].CSDN,2021-10-03.https://blog.csdn.net/shandianfengfan/article/details/120600453.

[30] 张海龙,袁国忠.C++_Primer Plus 中文版[M].北京:人民邮电出版社,2012.07.

[31]Anthony Williams.C++ Concurrency in Action[M].2012.1.

[32]Wikipedia.系统测试[J/OL].Wikipedia,2023-12-01.https://zh.wikipedia.org/wiki/%E7%B3%BB%E7%BB%9F%E6%B5%8B%E8%AF%95