关于 OpenHarmony 相机模块的图像质量优化的研究

路秉鑫
软件工程
3220241564@bit.edu.cn

石凯铭
软件工程
3220241575@bit.edu.cn

孙胤儒
软件工程
3220241581@bit.edu.cn

乌世杰
软件工程
3220241595@bit.edu.cn

摘要—本文详细探讨了 OpenHarmony 平台上相机 API 的设计与实现，重点分析了其技术架构、核心组件及功能特性。文章首先介绍了 OpenHarmony 相机 API 的基本工作原理，包括与底层操作系统（如 Linux 和 LiteOS）的交互机制以及其在多设备协同场景中的应用优势。随后，文章对比了 OpenHarmony 与 Android 平台在相机应用开发中的异同，突出了 OpenHarmony 在轻量化和性能优化方面的优势。除此之外，本文还深入分析了图像处理中的锐化算法原理与应用，通过实现非掩模锐化（USM）算法提升图像质量。最后，文章通过实验验证了锐化算法的有效性，为开发者提供参考。

关键词—OpenHarmony，相机 API，图像锐化，图像质量

I. 引言

在当今数字化时代，相机功能已成为智能手机、平板电脑、智能家居设备等各类智能终端不可或缺的重要组成部分。它不仅满足了用户日常拍照、录像的基本需求，还通过与人工智能、增强现实等技术的融合，为用户带来了更加丰富和智能的体验。例如，用户可以通过手机相机实现即时翻译、物体识别、虚拟试衣等功能，极大地拓展了相机的应用场景和价值。因此，相机模块的性能和功能直接关系到智能设备的用户体验和市场竞争力。

随着技术的不断进步，相机模块的应用场景也在不断扩展。在智能手机领域，高分辨率拍照、超广角镜头、微距拍摄、夜景模式等功能已经成为标配，用户对图像质量和拍摄体验的要求越来越高。在智能家居领域，智能摄像头不仅用于安全监控，还通过人脸识别、行为分析等功能，为用户提供更加智能化的家居管理解决方案。在工业领域，相机模块被广泛应用于质量检测、自动化生产等环节，对图像的精确度和处理速度提出了更高的要求。在医疗领域，内窥镜、显微镜等医疗设备中的相机模块，对图像的清晰度和稳定性有着极高的标准。这些应用场景的多样化和复杂化，对相机模块的设计和实现提出了更高的挑战。

在相机技术的发展过程中，图像预处理是一个关键环节。图像预处理是指在图像采集后，对图像进行一系列的处理操作，以提高图像的质量、增强图像的特征、减少噪声等，从而为后续的图像分析和应用提供更好的基础。常见的图像预处理操作包括曝光调整、白平衡校正、对比度增强、噪声去除、锐化等。这些操作对于提升图像的视觉效果和后续处理的准确性至关重要。例如，在低光照条件下拍摄的图像通常会存在曝光不足、噪点较多的问题，通过图像预处理可以显著改善这些问题，使图像更加清晰、明亮。此外，图像预处理还可以为后续的高级图像处理任务，如目标检测、图像分割、特征提取等，提供更加准确和可靠的数据基础。图像预处理技术的发展呈现出以下几个主要趋势：

1) 智能化处理：随着深度学习技术的发展，图像预处理将会更加智能化和自动化，以便于模型更好地学习和识别图像中的特征。例如，利用深度学习算法可以自动调整图像的曝光、对比度和色彩平衡，使图像更加自然和美观。
2) 实时处理：随着硬件性能的提升，图像预处理能够在更短的时间内完成，为用户提供即时的反馈。这对于实时视频应用，如视频会议、直播等，尤为重要。
3) 多模态融合：结合多种传感器数据，如光流传感器、深度传感器等，实现更全面的图像预处理效果。例如，通过光流传感器可以检测图像中的运

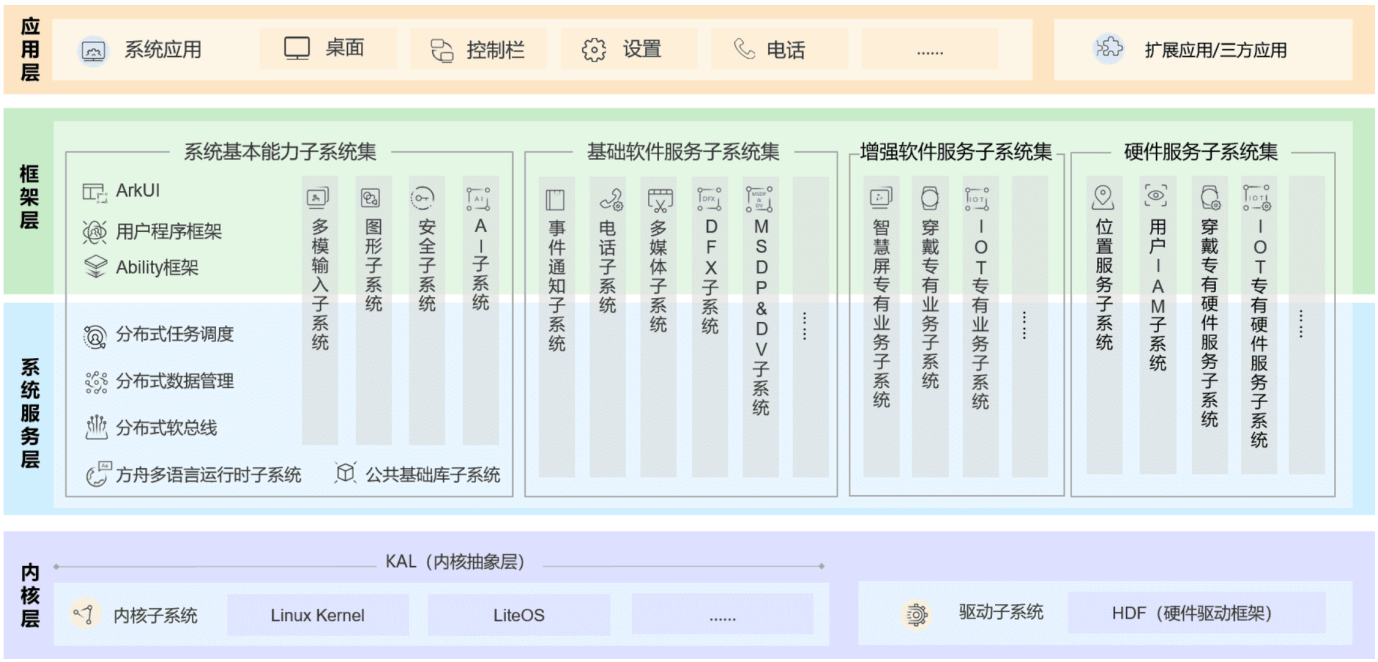

图 1. OpenHarmony 的技术架构.

动信息，从而实现更加精准的图像稳定和运动补偿。

4) 低功耗设计：随着设备续航时间的重要性日益凸显，如何在保证性能的同时降低功耗成为关键。例如，通过优化算法和硬件加速，减少图像预处理过程中的计算资源消耗，延长设备的续航时间。

在智能手机领域，相机功能的扩展也取得了显著进展。例如，Android 的 Camera2 和 CameraX API 提供了丰富的扩展功能，如自动模式、焦外成像、美颜、HDR（高动态范围）和夜间模式等。这些扩展功能可以根据当前场景自动调整相机参数，提升图像质量。

此外，智能相机在安防监控、医疗成像、智能交通等领域的应用也日益广泛。在安防监控中，智能相机通过人脸识别、行为分析等功能，提高了监控效率和安全性能。在医疗成像中，智能相机有助于提高诊断准确性和效率，如病理切片分析、手术导航等。在智能交通中，智能相机用于车牌识别、车辆检测等，提高了交通安全和效率。

OpenHarmony 作为一款面向全场景的开源操作系统，其独特的分布式架构和组件化设计，使其在万物互联的时代具有广阔的应用前景。OpenHarmony 通过组件化和组件弹性化等设计方法，做到硬件资源的可大可小，在多种终端设备间，按需弹性部署，全面覆盖了各种 CPU，从百 KiB 到 GiB 级别的 RAM。这种设计使得 OpenHarmony 能够灵活地适应不同形态的终端设备，满足不同场景下的应用需求。例如，在智能家居场景中，多个智能摄像头可以协同工作，实现全景监控和智能联动；在工业生产中，多个相机模块可以协同完成复杂的产品检测任务，提高生产效率和质量。OpenHarmony 的这种灵活性和适应性，为相机模块的多设备协同拍摄和数据共享提供了强大的支持，使得相机模块能够在不同设备之间实现更丰富的功能和更高效的性能。

OpenHarmony 的整体技术架构遵从分层设计，从下向上依次为内核层、系统服务层、框架层和应用层。内核层采用多内核（Linux 内核或者 LiteOS）设计，支持针对不同资源受限设备选用适合的 OS 内核。内核抽象层（KAL，Kernel Abstract Layer）通过屏蔽多内核差异，对上层提供基础的内核能力，包括进程/线程管理、内存管理、文件系统、网络管理和外设管理等。驱动子系统提供统一外设访问能力和驱动开发、管理框架，是系统硬件生态开放的基础。这种分层架构设计不仅提高了系统的可移植性和可扩展性，还为开发者提供了更加灵活和高效的开发环境，使得开发者可以更加专注于应用层的开发，而无需过多关注底层硬件的细节。

然而，尽管 OpenHarmony 相机模块已经具备一定

的功能和性能，但随着用户对相机功能要求的不断提高，以及智能设备应用场景的日益复杂，OpenHarmony 相机模块在图像预处理方面仍面临着诸多挑战和改进空间。例如，如何在保证图像质量的前提下，进一步提高图像预处理的速度和效率；如何更好地支持多摄像头协同工作，实现更丰富的图像预处理效果；如何优化图像预处理算法，减少计算资源的消耗，延长设备的续航时间等。这些问题的解决不仅需要对现有相机模块进行深入分析和研究，还需要结合最新的技术发展趋势，提出创新性的改进方案。

为了提升 OpenHarmony 相机模块的图像预处理性能和功能，本论文将首先对 OpenHarmony 相机组件架构进行详细分析，探讨其实现原理。其次，将对比分析 Android 操作系统的相机模块，探讨它们在图像预处/理功能丰富度、开发体验、跨平台适配等方面的表现，以及它们在解决上述挑战时所采用的技术手段和策略。通过对比分析，揭示 OpenHarmony 相机模块在图像预处理技术实现和应用效果上的异同，然后设计在应用层应用锐化处理的方法以提升图像质量。

本论文的研究不仅有助于提升 OpenHarmony 相机模块的图像预处理性能和功能，还将为开发者在选择和使用不同操作系统相机模块时提供决策依据，推动 OpenHarmony 在智能设备相机功能领域的进一步发展，为用户提供更加优质和智能的拍摄体验。通过本论文的研究，我们期望能够为 OpenHarmony 相机模块的图像预处理优化提供有价值的参考和借鉴，促进其在万物互联时代的广泛应用和持续发展。

II. 相关研究

A. *OpenHarmony* 相机组件

OpenHarmony 相机组件 [1] 为开发者提供了丰富的接口和强大的相机硬件控制能力，支持多种相机业务的开发，极大地简化了相机硬件的访问、操作以及新功能的开发。开发者可以通过统一的接口访问和操作相机硬件，灵活地控制相机设备，进行拍照、录像、预览等操作，满足不同场景下的相机功能需求。

OpenHarmony 相机组件从上到下依次可以分为：应用层-相机框架层-系统服务层-内核驱动层。如图 2所示。

1) 应用层: 在 OpenHarmony 中，应用层是用户与系统交互的最上层，主要通过使用 JS（JavaScript）或

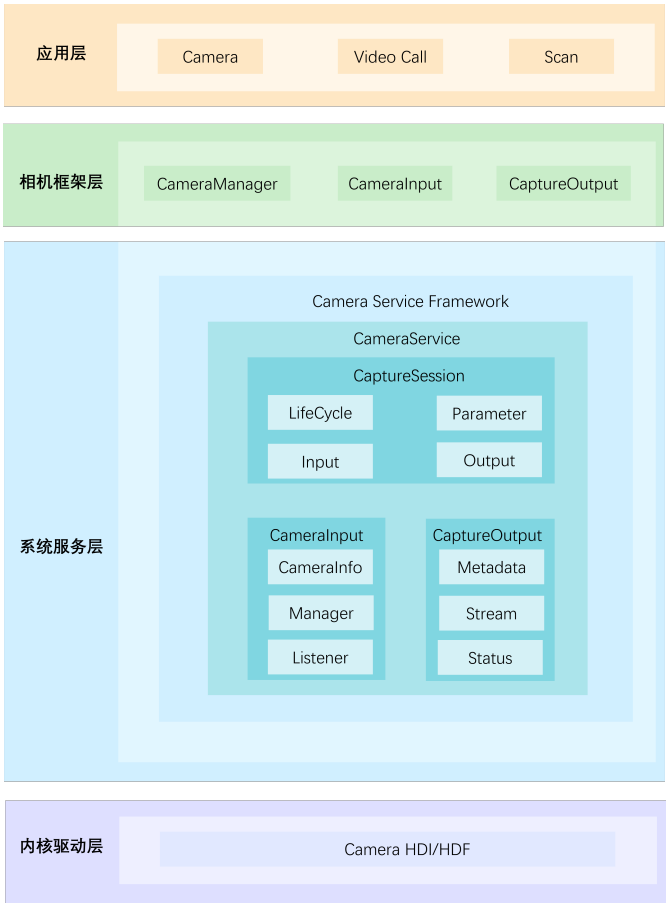

图 2. OpenHarmony 相机架构图

eTS（Enhanced TypeScript）语言进行开发。应用层的相机功能通常是通过调用系统提供的相机 API 实现的，这些 API 封装了底层硬件的操作，使得开发者无需关心具体的硬件实现即可使用相机功能。应用层与相机框架层的交互是通过 NAPI（Native API）[2] 接口完成的，NAPI 为应用层提供了一组便捷的调用接口，允许开发者以较高层次的抽象来操作相机，如拍照、录像、实时预览等。应用层和相机框架层之间通过 IPC（进程间通信）进行数据和指令的传递，IPC 保证了不同进程之间的安全、可靠的通信。

应用层主要关注于用户体验和上层功能的实现，开发者通过简洁的编程接口和高层的功能模块，能够实现复杂的相机操作，且无需深入了解硬件层的实现细节。这种层次化的架构大大提升了 OpenHarmony 在多设备、多平台环境下的应用开发效率，使得相机功能能够跨平台、跨设备地快速部署和实现。

2) 相机框架层: 在 OpenHarmony 的相机框架层，系统将相机功能抽象为三个核心模块：会话管理、设备输入和数据输出，这些模块通过相应的类进行实现。

会话管理主要负责相机采集的生命周期、参数配置、输入管理和输出管理，确保相机操作的有序进行。相机框架中的 CaptureSession 类便是会话管理的核心，它管理着相机的整个生命周期，并负责配置相机的工作参数、设备输入和数据输出。设备输入模块通过 CameraInput 类对相机设备进行管理，负责设备的查询、控制和监听，确保相机能够根据需求进行适当的输入设置，例如调整闪光灯模式、设置曝光值、选择焦点等。设备输入的设置确保了相机设备能够根据不同的场景和需求动态调整其输入参数。数据输出模块则通过 CaptureOutput 类实现数据的输出管理。CaptureOutput 根据具体的业务需求进一步细分为三种不同的输出类型：PreviewOutput、PhotoOutput 和 VideoOutput，分别对应相机的预览输出、拍照输出和录像输出。每一种输出类型负责不同的数据流处理，预览输出主要用于实时显示相机捕捉的画面，拍照输出则用于获取静态图片数据，而录像输出则处理视频流的捕获与存储。所有这些类都由相机管理类 CameraManager 进行统一管理和调用，CameraManager 是整个相机框架的入口，负责协调会话管理、设备输入和数据输出之间的交互与通信。

3) 系统服务层: 在 OpenHarmony 的系统服务层，CameraService 扮演着至关重要的中间人角色，负责在相机应用（camera app）和相机主机服务（camera host）之间传递信息，确保相机功能的正常调用和消息反馈。具体来说，CameraService 实现了多个接口，通过这些接口接收来自应用层的相机操作请求，并将其传递给相机主机服务。与此同时，CameraService 也负责接收相机主机服务的反馈消息，并将这些消息转发给应用层，保证系统中不同组件之间的通信与协作。这个过程主要通过 IPC（进程间通信）机制完成，使得不同进程间能够安全高效地进行数据交换和命令传递。CameraService 的设计使得应用层无需直接与底层硬件打交道，而是通过统一的接口调用方式，轻松实现对相机的操作。这种设计不仅保证了高层与低层的解耦，还提高了系统的可维护性和可扩展性。

CameraService 的实现包含了多个关键接口，主要涉及相机设备的管理、会话的控制、参数配置的设置、输入输出的配置等。应用层通过调用这些接口发起对相机设备的管理请求，包括启动、停止、配置参数、获取实时预览数据等。CameraService 将这些请求传递给相机主机服务，后者根据具体的硬件和操作系统环境来执行相应的操作，并将执行结果反馈给 CameraService，再由其将结果返回给应用层。通过这种中介式的消息转发机制，CameraService 确保了各个组件之间的协作更加灵活高效。

4) 内核驱动层: OpenHarmony 相机组件在内核驱动层采用三层架构：HDI 实现层、框架层和设备适配层。该分层设计使得相机驱动框架既高效管理硬件设备，又能兼容多平台和芯片差异，同时提供统一的南向接口和数据流转发功能。

HDI 实现层负责实现标准的硬件设备接口（HDI），为上层框架提供统一的南向接口。此层直接控制硬件设备，包括相机的初始化、启动、停止、参数配置等操作，确保相机设备的生命周期管理标准化。框架层位于 HDI 层之上，负责接收上层指令并转发到 HDI 层进行实际的硬件操作。它实现了相机数据流的转发和处理，并管理各硬件设备的协调工作。在不同模式下（如拍照、录像、预览），框架层根据配置调整数据流路径，确保相机功能模块能够动态变化，支持实时预览、图像处理或视频录制等功能。设备适配层负责屏蔽平台和芯片之间的差异，为 OpenHarmony 提供跨平台的硬件适配能力。它通过抽象硬件差异，使系统能够在多种硬件平台和操作系统环境下无缝运行，提升系统的可扩展性和可移植性。

整体设计使得相机硬件的管理更加模块化，HDI 层标准化硬件接口，框架层管理数据流和硬件设备，设备适配层确保跨平台兼容性。这种架构提升了相机驱动框架的高效性、稳定性、可维护性和扩展性，为开发者提供了标准化、高效的相机接口，简化了应用开发中的相机控制复杂性，同时确保了相机在不同设备上的兼容性和性能。

B. 安卓系统相机设计

安卓操作系统作为全球最广泛使用的移动平台之一，其相机框架经历了显著的发展历程。早期使用简单的 Legacy Camera API，但其灵活性和性能逐渐受限，无法满足高级需求（如精细手动控制和 RAW 数据）。为解决此问题，Android 引入了 Camera2 API，提供更

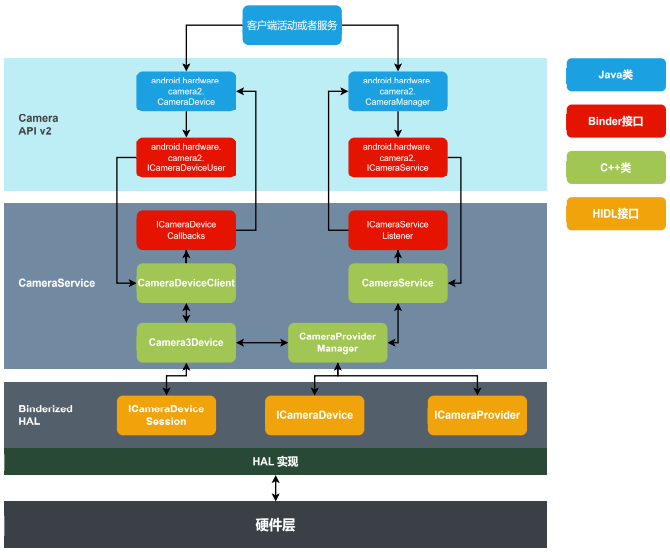

图 3. 安卓相机架构图

精细的硬件控制，支持高级功能。然而，Camera2 API 复杂性较高。为简化开发并解决兼容性问题，谷歌推出了 CameraX 库 [3]，它基于 Camera2 提供更高层次的抽象，使常见相机用例的实现更简单可靠。

1) 架构介绍: 图 3展示了在 Android 8.0 及以上版本的安卓系统中的基于 Camera2 的相机架构 [4]。该架构是一个多层次、协同工作的系统，旨在实现应用程序与相机硬件之间的有效沟通和控制。从用户的角度来看，他们通过应用与相机进行交互，而底层则涉及多个组件和进程的协作。从上向下，可分为以下的几个层次:

- 应用层 (Camera App): 位于整个架构的最顶端，直接与用户交互。该层负责接收用户的直接操作，并将其转化为具体相机需求，然后通过 Camera2 API 标准接口发送需求，等待处理结果。
- Camera2 API 层: 以 JAR 包的形式存在于 App 进程中，负责处理进程间的通信。该层封装了 Camera2 API 具体实现细节，暴露功能接口以接收应用层的请求，同时维护内部的业务逻辑，然后通过调用 Camera AIDL（Android Interface Definition Language）跨进程接口，将请求发送到独立的 Camera Service 进程进行处理。
- Camera Service 层: 作为一个独立的进程存在于 Android 系统中，负责协调和管理不同的相机设备，并作为中间层连接上层框架和底层硬件抽象层。该层封装了 Camera AIDL 跨进程接口，供 Camera2 API 层调用，接收来自应用框架的图像请求，维护着请求的处理逻辑，通过调用 Camera HIDL（HAL Interface Definition Language）跨进程接口，将请求进一步下发到独立的 Camera Provider 进程中。
- Camera HAL 层: 作为一个独立的进程存在于 Android 系统中。它封装了 Camera HIDL 功能实现，将跨进程接口提供 Camera Service 进行调用，接收来自 Service 的图像请求，并在内部加载由 OEM/ODM（原始设备制造商/原始设计制造商）实现的 Camera HAL Module，通过 Camera HAL3 接口控制底层的传感器、控制 ISP 等 Camera Hardware 部分，等待处理完毕后将结果发回 Camera Service 层。
- 硬件层: 是相机系统的物理实现部分，通常包括镜头、传感器、ISP(Image Signal Processor) 等重要的模块，还有对焦马达、闪光灯、滤光片、光圈等辅助模块。

2) 优点分析: 安卓相机的架构设计具有以下的优点:

清晰的分层结构与明确的职责划分: Android Camera 架构采用了清晰的分层设计，从应用层到硬件驱动层，每一层都承担着特定的职责。各层之间通过标准接口进行通信，显著降低了层与层之间的耦合度，使各层相对独立，显著简化了维护和升级工作。

灵活性与可定制性: 接口与实现分离的设计原则是该架构灵活性的关键。它确保了整体架构的稳定性，同时允许系统在细节实现上具有多样性，体现了强大的灵活性。例如，CameraX 允许同时使用 Preview、VideoCapture、ImageAnalysis 和 ImageCapture 这四种用例，并且每个用例都可以独立使用。应用层可以根据具体的业务需求灵活配置执行流程和参数，从而实现高度定制化的拍摄体验。

可靠隐私安全性: 所有用户都不希望基于 Android 的应用程序窃取自己的私人数据，而常规的进程通信不安全，很容易被各种木马劫持利用。木马可以轻易地推送程序的接收地址以建立连接，这几乎是不可能阻止的 [5]。Binder 是在这种背景下出现的一种新机制，他的驱动程序运行在内核空间上 [6]，并采用 C/S 通信模式，其中一个进程作为服务器 [7] 提供一些系统服务。其他进程作为客户端，向服务器发送服务请求，由业务管理单元统一调度，提供相关的系统服务。传输过程只需要

一个数据副本，并且数据发送方可以识别用户和进程的 ID，因此传输效率和安全性能都得到了提高。在 HAL 层安卓相机充分利用了 Binder 的优点，保障了用户在使用相机时的隐私安全。

强大的硬件兼容性: Android 系统需要适配众多手机制造商的不同硬件配置和传感器型号。通过分层设计，无论底层硬件如何更换，Camera HAL 层及其上层的代码通常无需进行大幅修改，即可在各种配置的设备上顺利运行。这种设计极大地提高了系统的硬件兼容性和可移植性，降低了开发和维护成本。

优化的性能表现: Camera2 的操作基于请求队列和异步回调机制。应用程序发送请求后，无需阻塞等待，而是通过监听回调来获取操作结果，显著提升了相机操作的效率。

3) 缺陷分析: 尽管如此，安卓相机架构设计也存在一些缺陷，具体表现在:

较高的学习门槛: Camera2 架构相对复杂，其使用方式不像传统的线性调用那样直观，开发者需要深入理解多个回调的使用流程和状态管理。

设备适配的差异性挑战: 由于不同设备制造商 (OEM) 对 Camera HAL 层的实现可能存在显著差异，导致在使用 Camera 架构时，在不同设备上可能会遇到适配性问题。这给开发者进行多设备适配带来了挑战，需要进行额外的测试和兼容性处理。

总之，安卓相机架构在分层设计、灵活性和兼容性方面表现出色，并提供了优化的性能潜力。然而，其复杂性导致了较高的学习成本，并且在某些特定场景下仍然存在功能限制和设备适配的挑战。

III. 方法设计

在之前的章节中，我们详细阐述了 OpenHarmony 相机系统与安卓相机系统的架构，从底层的硬件抽象层到核心的框架层，最终延伸至用户直接交互的应用层，各个模块紧密协作，共同完成了图像的采集、初步处理和呈现。这一系列流程确保了相机能够捕捉到现实世界的影像，并进行诸如自动曝光、自动对焦、自动白平衡等基础优化。然而，为了追求极致的图像质量，并为用户带来更出色的拍摄体验，我们认识到，仅仅依靠系统底层的初步处理是不够的。因此，我们决定在应用层引入图像锐化 (Sharpening) 功能。本章将探讨在 OpenHarmony 相机应用层实现自动图像锐化的策略、核心技术以及具体的实现细节，旨在阐明如何通过应用层的优化，显著提升照片的清晰度和信息量。

虽然 OpenHarmony 系统底层和框架层已经具备了诸如自动曝光、自动对焦、自动白平衡等基础的图像处理能力，这些处理为成像奠定了良好的基础。然而，我们必须承认，受限于硬件本身的光学特性、传感器性能以及系统层面的通用性考量，这些基础处理往往无法完全满足用户对于高质量照片的期望。例如，镜头固有的像差可能导致图像边缘模糊，传感器在捕捉细节时可能存在一定的损失，而系统层面的降噪算法在去除噪点的同时，也可能平滑掉一些细微的纹理。因此，即使经过了这些初步处理，照片仍然可能显得不够锐利，细节不够突出。

图像锐化技术，正是在这一背景下显得尤为有效。其核心目标在于有选择地增强图像中不同区域之间的对比度，特别是边缘和纹理细节的对比度，从而在视觉上提升图像的清晰度和锐利度。适度的锐化处理能够“点石成金”，让模糊的边缘变得清晰，让丢失的细节重现，赋予照片更强的视觉冲击力和信息承载能力。尤其是在光线条件不佳，或者使用了并非旗舰级的相机模组时，锐化能够有效地弥补硬件上的不足，显著提升照片的观感。引入应用层锐化，能补系统底层处理的局限性，提升整体的用户体验。

目前，常用的锐化算法包括拉普拉斯算子 (Laplace Operator) [8]、非锐化掩模 (Unsharp Masking, USM) [9]、高提升滤波 (High-Boost Filtering) [10]、索贝尔算子 (Sobel Operator) [11]、Prewitt 算子 (Prewitt Operator) [12] 等，这些算法各有优缺点，如拉普拉斯算子对噪声敏感，高提升滤波虽可增强锐度但也可能放大噪声，而 Sobel 和 Prewitt 算子则侧重边缘检测，难以直接锐化。

因此我们将直接运用效果与速度均优秀的 USM 算法来进行锐化，该算法核心思想是从原始图像中减去一个模糊版本，得到一个包含高频信息的 “掩模”，然后将这个掩模以一定的权重加回到原始图像中，从而增强图像的边缘和细节，达到锐化的效果。算法流程如下:

首先，对原始图像 $I(x,y)$ 进行模糊处理 (Blurring)，得到模糊后的图像 $I_blur(x,y)$。常用的模糊方法是应用高斯滤波器，对应的高斯滤波器核 $G(x,y,\sigma)$ 的定义如公式 (1) 所示:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

其中，σ 是标准差，控制模糊的成都。模糊后的图像可以通过公式 (2) 的卷积运算得到:

$$I_{blur}(x,y) = I(x,y) * G(x,y,\sigma) \tag{2}$$

这里的 $*$ 表示卷积操作。

接下来则要创建淹模 (Masking)，通过公式 (3) 从原始图像中减去模糊后的图像，得到非锐化掩模 $M(x,y)$:

$$M(x,y) = I(x,y) - I_{blur}(x,y) \tag{3}$$

这个掩模主要包含了图像的高频信息，也就是边缘和细节部分。

最后进行图像的锐化，通过公式 (4) 将掩模 $M(x,y)$ 乘以一个放大因子 α，然后加回到原始图像 $I(x,y)$ 中，得到锐化后的图像 $I_sharp(x,y)$:

$$I_{\text{sharp}}(x,y) = I(x,y) + \alpha \cdot M(x,y) \tag{4}$$

将掩模的定义代入上式，可以得到 USM 算法的完整公式 (5):

$$I_{\text{sharp}}(x,y) = I(x,y) + \alpha \cdot (I(x,y) - I_{blur}(x,y)) \tag{5}$$

USM 算法具有原理简单，实现方便，并且能够有效地增强图像的边缘和细节优点，在图像处理软件和系统中被广泛应用，例如 Adobe Photoshop 中就实现了该锐化算法。但是 USM 算法可能放大噪声，因为 USM 算法增强的是高频信息，如果图像本身含有噪声，锐化过程也会放大这些噪声，导致图像质量下降。

IV. 实验

在本章中，我们将通过一系列实验来验证前述在应用层引入图像锐化算法的有效性。通过对处理前后照片的对比分析，我们将初步评估所选锐化算法在提升图像清晰度和细节表现方面的能力。

A. 实验设置

考虑到本次小论文的完成时间和资源成本的限制，我们暂不直接在真实的 OpenHarmony 设备上进行完整的应用集成和测试。为了更高效地验证锐化算法的核心逻辑和效果，我们将选择在一个可控的、便捷的环境下进行初步的实验评估。具体而言，我们将在 Windows 操作系统平台上，利用 C 语言编写的图像处理源代码，

图 4. 原图 (左) 与锐化后 (右) 对比

对实际拍摄的照片进行锐化处理，并观察其效果。相关代码已同步到远程仓库中[1]。这样的实验设计能够使我们集中精力评估算法本身的处理能力和视觉效果，简化了环境配置和调试的复杂性，为在 OpenHarmony 平台上的部署提供了理论和实践基础。

1) 数据集: 在实验中，我们使用了公开的鸟类图像数据集 CUA-200-2011 [13]。该数据集作为 CUB-200 的扩展，包含 200 个鸟类类别，共计 11,788 张图像，并广泛应用于小样本细粒度图像分类及检测任务的相关研究中。本研究随机选取了其中的 200 张图像作为锐化处理的对象。

2) 指标: 在实验中，我们使用 Tenengrad 函数 [14] 来衡量图像清晰度，其核心思想是通过计算图像梯度强度来评估图像的锐利程度。清晰的图像通常具有更明显的边缘和细节，对应着更大的梯度值。Tenengrad 函数通过计算图像中像素点梯度平方和的平均值来量化这种清晰度。其简要公式如下:

$$\text{Score} = \frac{1}{M \text{ imes } N} \sum_{x=1}^{M} \sum_{y=1}^{N} \left(G_x^2(x,y) + G_y^2(x,y) \right) \tag{6}$$

$$G_x(x,y) = I(x,y) * K_x \tag{7}$$

$$G_y(x,y) = I(x,y) * K_y \tag{8}$$

其中 $I(x,y)$ 是原始图像，而 $G_x(x,y)$ 与 $G_y(x,y)$ 则表示水平和垂直方向的梯度，$*$ 代表卷积运算，K_x 与 K_y 则为卷积核。

B. 案例研究

图 4 为一个典型的案例研究，直观地展示了我们所实现的 USM 算法在图像锐化方面的有效性。左侧为原始图像，右侧为经过算法处理后的图像。对比两图，

[1]https://github.com/ShahidImae/OH_sharpening

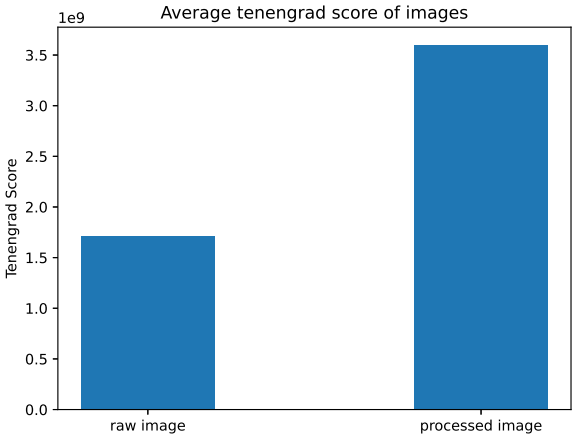

图 5. Tenegrad 得分

可以明显看出，经过锐化处理后的图像，其细节信息得到了有效增强，图像的清晰度得到了显著提高。这充分说明了我们实现的 USM 算法在提升图像质量方面的作用。

C. 实验结果

为了更充分地展现我们算法在图像锐化方面的实际效果和可靠性，我们进行了一项细致的实验评估。具体而言，我们从随机选择的数据集中抽取了 200 张具有代表性的图片，并对这些图片逐一应用了我们所实现的锐化算法。为了量化锐化效果，我们分别计算了原始图像以及锐化后图像的 Tenegrad 函数得分。考虑到单张图片结果的偶然性，我们对这 200 张图片计算得到的 Tenegrad 得分分别取了平均值，以获得更具统计意义的结论。

实验结果如图 5 所示。显然，经过我们的算法处理后，图片的 Tenegrad 平均得分相较于原始图像提升至原来的两倍。这一显著的提升有力地证明了我们算法在增强图像清晰度方面的卓越性能和有效性。

总结

本文探讨了新兴的 OpenHarmony 操作系统与成熟的 Android 系统在相机框架上的异同与融合策略。通过细致地剖析 OpenHarmony 的模块化组件结构和数据处理流程，我们清晰地揭示了其在跨平台兼容性和性能优化方面的内在优势。OpenHarmony 凭借其分布式架构和更精简的设计，能够更好地适应不同硬件环境，并在资源受限的设备上表现出更高的效率。相比之下，Android 平台虽然拥有庞大而成熟的生态，但在相机 API 的实现层面，仍然存在一定的复杂性，并且在某些场景下，资源消耗相对较高，尤其是在处理高分辨率图像或进行复杂图像处理时。

在图像处理能力方面，本文选取了具有代表性 USM 锐化算法进行图像质量提升实验，实验数据明确地表明，该算法都能够有效地增强图像的细节和清晰度，显著提升视觉观感，从而为各类多媒体应用，例如移动摄影、视频编辑、以及新兴的 AR/VR 应用等，提供更为优质且沉浸式的用户体验。这突显了图像处理技术在现代移动设备中的重要性。

随着 OpenHarmony 生态的不断发展壮大，未来研究还可以关注如何更好地实现与 Android 生态的兼容与协同，例如在相机框架层面实现更高效的数据互通和功能迁移，从而为开发者提供更灵活、更强大的工具和平台。总而言之，本文的研究不仅加深了我们对 OpenHarmony 和 Android 在相机框架上的理解，也为未来的技术发展和应用创新奠定了基础。我们相信，随着技术的不断进步和研究的深入，移动设备的图像处理能力将迎来更加辉煌的未来。

参考文献

[1] (2024) Openharmony camera. [Online]. Available: https://gitee.com/openharmony/multimedia_camera_standard

[2] (2024) Openharmony napi. [Online]. Available: https://gitee.com/openharmony/arkui_napi

[3] (2024) Camerax overview. [Online]. Available: https://developer.android.com/media/camera/camerax

[4] (2024) Asop camera. [Online]. Available: https://source.android.google.cn/docs/core/camera

[5] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Drebin, "Effective and explainable detection of android malware in your pocket," in *Network and distributed system security symposium*, 2014, pp. 1–15.

[6] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, 2014, pp. 123–134.

[7] A. P. Bradley, "The use of the area under the roc curve in the evaluation of machine learning algorithms," *Pattern recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.

[8] T. Ma, L. Li, S. Ji, X. Wang, Y. Tian, A. Al-Dhelaan, and M. Al-Rodhaan, "Optimized laplacian image sharpening algorithm based on graphic processing unit," *Physica A: Statistical Mechanics and its Applications*, vol. 416, pp. 400–410, 2014.

[9] Z. Al-Ameen, A. Muttar, and G. Al-Badrani, "Improving the sharpness of digital image using an amended unsharp mask filter." *International Journal of Image, Graphics & Signal Processing*, vol. 11, no. 3, 2019.

[10] Y. Zhang, X. Tian, and P. Ren, "An adaptive bilateral filter based framework for image denoising," *Neurocomputing*, vol. 140, pp. 299–316, 2014.

[11] G. Ravivarma, K. Gavaskar, D. Malathi, K. Asha, B. Ashok, and S. Aarthi, "Implementation of sobel operator based image edge detection on fpga," *Materials Today: Proceedings*, vol. 45, pp. 2401–2407, 2021.

[12] R.-G. Zhou, H. Yu, Y. Cheng, and F.-X. Li, "Quantum image edge extraction based on improved prewitt operator," *Quantum Information Processing*, vol. 18, pp. 1–24, 2019.

[13] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "Cub-200-2011," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

[14] L. Her and X. Yang, "Research of image sharpness assessment algorithm for autofocus," in *2019 IEEE 4th International Conference on Image, Vision and Computing (ICIVC)*, 2019, pp. 93–98.