程序总体框架设计报告

09021227 金桥

2023 年 10 月 2 日

1 实验目标

《数字图像处理》课程的后续实验包括：图像几何变换、图像灰度变换、图像增强、简易医学透视图像浏览器。本次实验为后续实验设计总体框架（为后续实验搭建一个合理的平台，方便逐步扩展功能，避免重复编写代码），并且完成灰度图像文件的读入和显示。

- 使用 C++ 程序设计语言以及集成开发环境 Qt
- **读取 bmp 格式的图像文件并完成傅里叶变换（傅里叶变换可调用现成的函数）**，但所设计的程序框架要允许增加非标准格式（自定义）的图像数据文件的方法
- **框架中应允许显示多幅图像**（例如，显示处理前的图像和处理后的图像）
- 总体框架着重考虑类的数据和方法的封装，以及类之间的关系

2 程序总体框架

程序的总体框架包括 App, Core, ImgProvider, FourierTrans 四个类。

2.1 App 类

App 类是整个程序的根本，连接后端图像处理以及前端页面展示。

```
1   class App: public QQuickView {
2       Q_OBJECT
3   public:
4       App(QWindow *parent = 0): QQuickView(parent), _imgCore(new
            Core()) {...}  // 构造函数，加载 UI 界面
5   public slots:
6       void loadImg() {...};            // 加载图像
7       void fourierTrans() {...}        // 调用 OpenCV 的傅里叶变换
8       void customFourierTrans() {...} // 调用自制的傅里叶变换
9       void quitApplication() {...}     // 用于退出程序的函数
10
11  private:
12      Core* _imgCore;                  // 后端处理核心
13  };
```

2.2 Core 类

Core 类存储后端处理图像用到的数据。

```
1   class Core {
2   public:
3       Core() = default;
4       // 获取图像文件的路径
5       [[nodiscard]] std::string getImgPath() const {...}
6       // 获取原图像内容
7       [[nodiscard]] cv::Mat getOriImgMat() const {...}
8       // 获取处理后图像的内容
9       [[nodiscard]] cv::Mat getDstImgMat() const {...}
10      // 设置处理后的图像
11      void setDstImgMat(cv::Mat img) {...}
12      // 从文件加载图像
13      void loadImg(std::string img_path) {...}
14
15  private:
16      std::string _imgPath;  // 图像文件路径
17      cv::Mat _oriImgMat;    // 原图像文件数据
18      cv::Mat _dstImgMat;    // 处理后图像数据
19  };
```

2.3 ImgProvider 类

ImgProvider 类用于为前端页面提供图像显示。

```
1   class ImgProvider: public QQuickImageProvider {
2   public:
3       ImgProvider(Core* imgCorePtr): QQuickImageProvider(
            QQuickImageProvider::Pixmap), _imgCore(imgCorePtr) {}
4       // 为前端页面提供 QPixmap
5       QPixmap requestPixmap(const QString &id, QSize *size, const
            QSize &requestedSize) override {...}
6
7   private:
8       // 将图像数据转换为前端显示的格式
9       QPixmap cvtMat2Pixmap(const cv::Mat& mat) {...}
10      Core* _imgCore;
11  };
```

2.4 FourierTrans 类

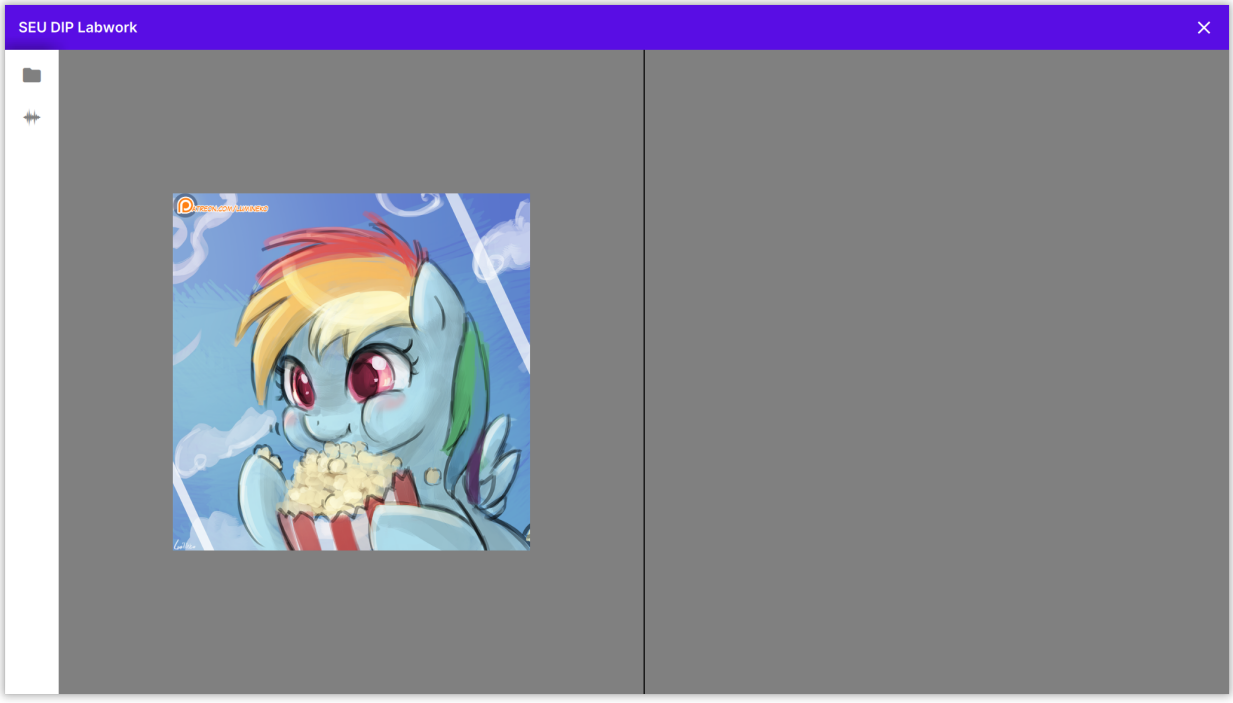
FourierTrans 类用于计算傅里叶变换。自制傅里叶变换原理参见第 4 节。

```
1    class FourierTrans {
2    public:
3        // 使用 OpenCV 实现傅里叶变换，速度较快
4        [[nodiscard]] cv::Mat fourierTrans(cv::Mat srcImage) {...}
5        // 自制傅里叶变换，速度较慢
6        [[nodiscard]] cv::Mat customFourierTrans(const cv::Mat &srcImg
             ) {...}
7    private:
8        const double _pi = 3.1415926535897932;
9    };
```

3 运行示例

以对图像进行傅里叶变换为例。

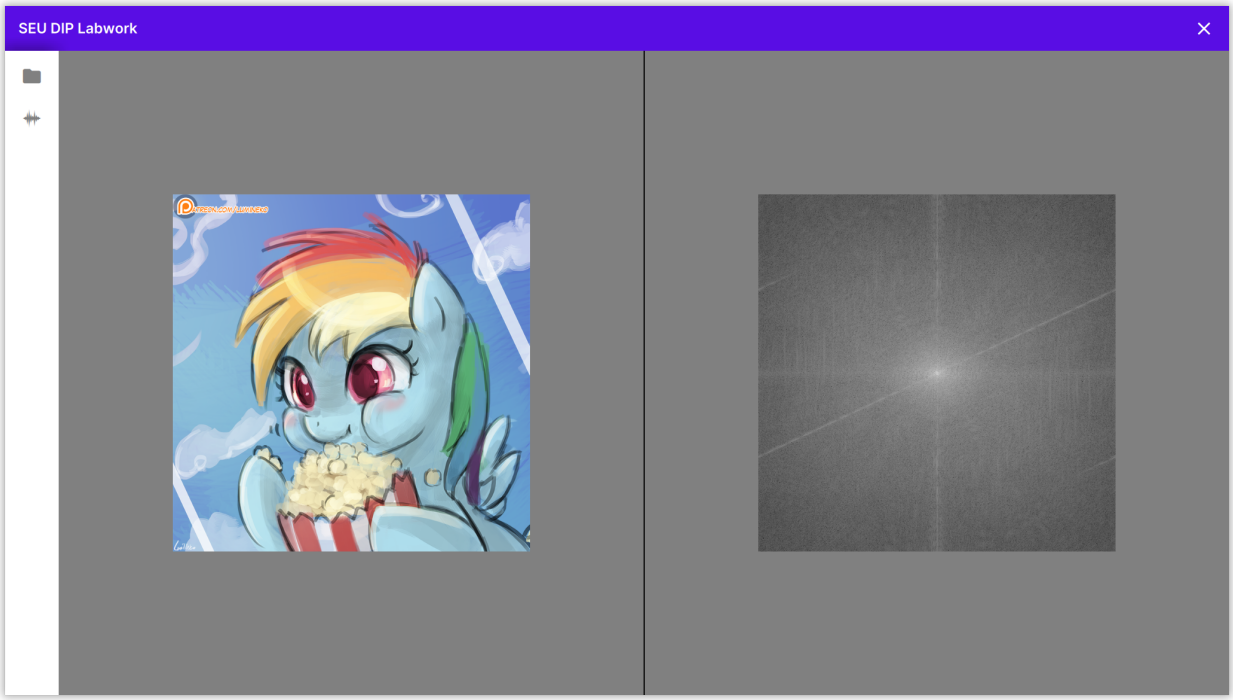
首先打开程序，点击左上角文件图标，选择 bmp 文件导入，导入之后如图所示。左侧展示的是未处理的图像，右侧是处理后的图像。



点击左侧波形图标，右侧即可出现经过傅里叶变换的图像。

- **左键点击**为调用 OpenCV 的傅里叶变换
- **右键点击**为调用自制的傅里叶变换

两侧的图像均可鼠标拖动并使用滚轮缩放查看。



4 傅里叶变换

4.1 使用 OpenCV 实现傅里叶变换

通过调用 OpenCV 提供的 `dft` 函数可以方便快速的实现傅里叶变换。

4.2 自制傅里叶变换

通过以下二维离散傅里叶变换公式可以实现一个简易的傅里叶变换：

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

首先将 `cv::Mat` 格式的数据转换为 `double` 类型的数组，之后进行两轮循环计算，每次计算再次遍历整张图像，时间复杂度为 $O(m^2n^2)$. 最后进行归一化并调整图像，使得低频位于中央即可。具体实现参照代码。