

Project 2 - 1 实验报告（简单图像处理程序）

- 学号 24344064
- 姓名 廖海涛

Project 2 - 1 实验报告（简单图像处理程序）

程序功能简要说明

程序运行截图

部分关键代码及其说明

命令解析

图像展示

灰度化

图像缩放

图像读写

图像压缩

程序运行方式

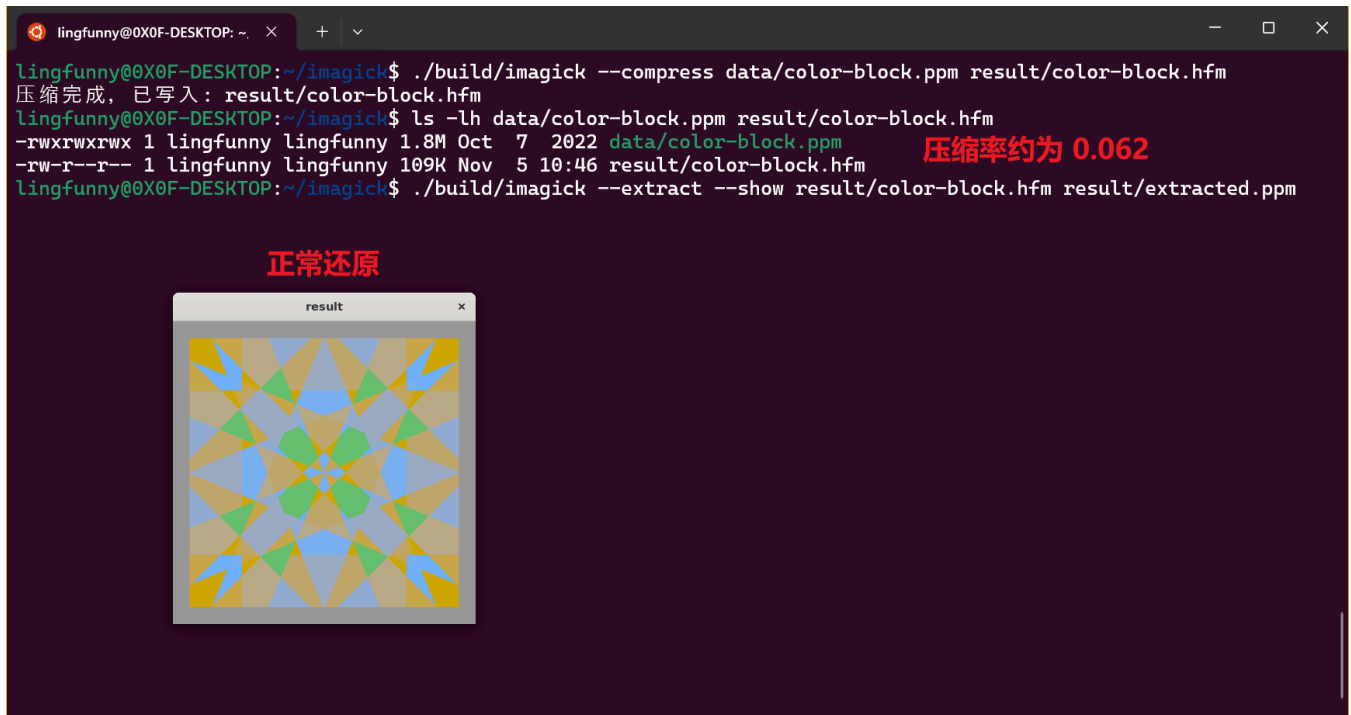
1. 程序功能简要说明

实现了类似命令行指令的解析，类似 ImageMagick。支持对图像的简单操作。

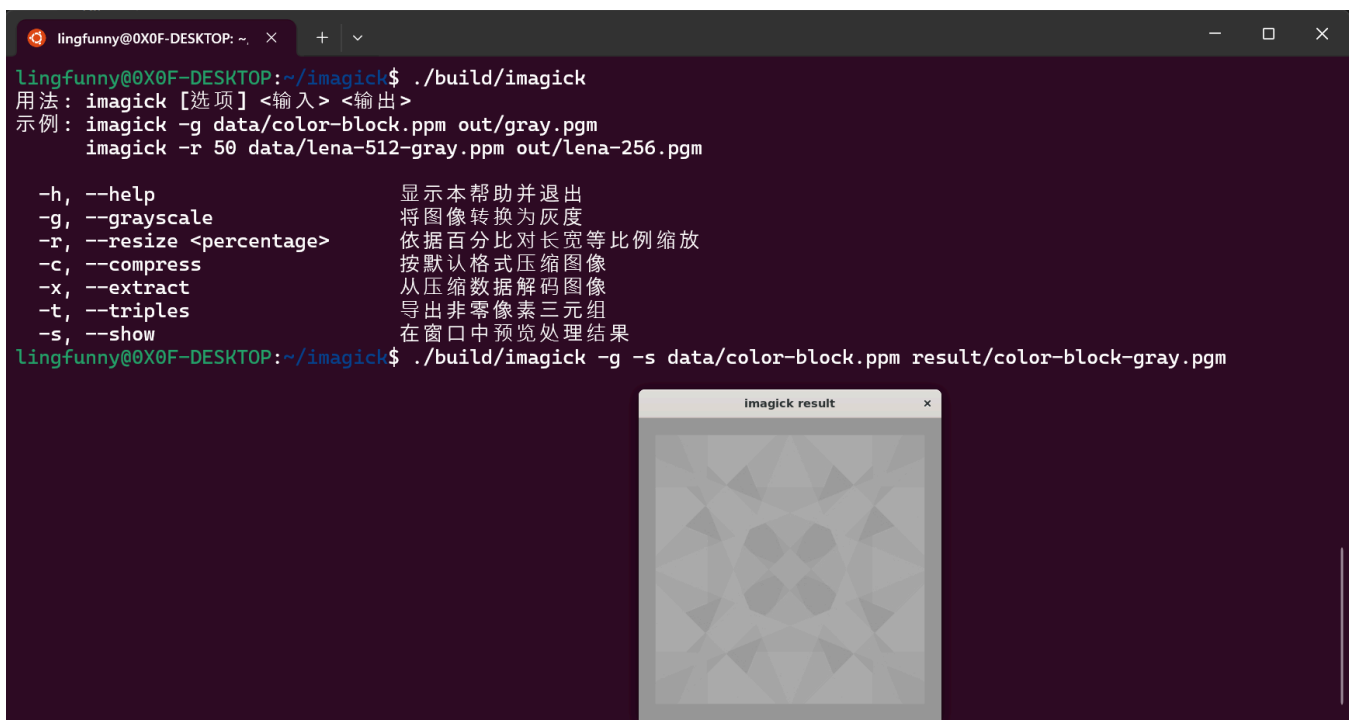
```
1  用法: imagick [选项] <输入> <输出>
2  示例: imagick -g data/color-block.ppm out/gray.pgm
3         imagick -r 50 data/lena-512-gray.ppm out/lena-256.pgm
4
5  -h, --help           显示本帮助并退出
6  -g, --grayscale      将图像转换为灰度
7  -r, --resize <percentage> 依据百分比对长宽等比例缩放
8  -c, --compress       按默认格式压缩图像
9  -x, --extract        从压缩数据解码图像
10 -t, --triples        导出非零像素三元组
11 -s, --show           在窗口中预览处理结果
```

2. 程序运行截图

压缩功能展示：



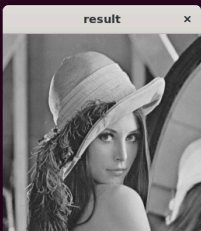
转为灰度图功能展示:



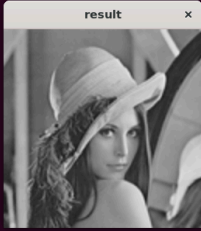
缩放功能展示:

```
lingfunny@0X0F-DESKTOP: ~$ ./build/imagick --resize 50% --show data/lena-512-gray.ppm result/lena-256-gray.ppm
处理完成, 已保存到: result/lena-256-gray.ppm
lingfunny@0X0F-DESKTOP:~/imagick$ ls -lh result/lena-256-gray.ppm
-rw-r--r-- 1 lingfunny lingfunny 237K Nov  5 10:49 result/lena-256-gray.ppm
lingfunny@0X0F-DESKTOP:~/imagick$ ./build/imagick --resize 200% --show data/lena-128-gray.ppm result/lena-256-gray.ppm
处理完成, 已保存到: result/lena-256-gray.ppm
lingfunny@0X0F-DESKTOP:~/imagick$ ls -lh result/lena-256-gray.ppm
-rw-r--r-- 1 lingfunny lingfunny 237K Nov  5 10:50 result/lena-256-gray.ppm
lingfunny@0X0F-DESKTOP:~/imagick$
```

512->256



128->256



3. 部分关键代码及其说明

3.1 命令解析

利用 `main(int argc, char** argv)` 函数的参数得到用户给出的参数。之后对于每个指令参数，逐一解析。对于非指令参数，视为文件参数。可以得到指令序列。

```
1  CLIConfig parseArguments(int argc, char** argv) {
2      if (argc <= 1) {
3          printUsage(std::cout);
4          std::exit(EXIT_SUCCESS);
5      }
6
7      CLIConfig config;
8      /*
9       * config.operations 存储操作序列
10     * config.inputPath 存储输入文件路径
11     * config.outputPath 存储输出文件路径
12     */
13     std::vector<std::string> positional;    // file path parameters
14
15     for (int i = 1; i < argc; ++i) {
16         std::string arg = argv[i];
17
18         if (arg == "--help" || arg == "-h") {
19             printUsage(std::cout);
20             std::exit(EXIT_SUCCESS);
21         }
22
23         if (!arg.empty() && arg[0] == '-') {
24             operationType type = parseOperationToken(arg);
25             std::string parameter;
```

```

26         if (operationRequiresArgument(type)) {
27             if (i + 1 >= argc) {
28                 throw std::runtime_error(arg + " 需要参数");
29             }
30             parameter = argv[++i];
31         }
32         config.operations.push_back({type, parameter});
33         continue;
34     }
35
36     positional.push_back(arg);
37 }
38
39 if (positional.empty()) {
40     throw std::runtime_error("请指定输入文件路径");
41 }
42 if (positional.size() > 2) {
43     throw std::runtime_error("请指定输入文件路径和输出文件路径");
44 }
45
46 config.inputPath = positional.front();
47 config.outputPath = positional.back();
48
49 return config;
50 }

```

3.2 图像展示

调用 opencv 库函数 `cv::imshow` 展示图像。需要注意 opencv 使用 BGR 通道顺序，而程序中使用 RGB 通道顺序存储彩色图像，因此需要进行转换。

```

1  void showImage(const cv::Mat& image, const std::string& windowTitle) {
2      if (image.empty()) {
3          throw std::runtime_error("无法展示空图像");
4      }
5      cv::Mat converted;
6      const cv::Mat* toDisplay = &image;
7      if (image.channels() == 3) {
8          cv::cvtColor(image, converted, cv::COLOR_RGB2BGR);
9          toDisplay = &converted;
10     }
11     cv::namedWindow(windowTitle, cv::WINDOW_AUTOSIZE);
12     cv::imshow(windowTitle, *toDisplay);
13     cv::waitKey(0);
14     cv::destroyWindow(windowTitle);
15 }

```

3.3 灰度化

在 [ImageOps.cpp](#) 中实现。调用 opencv 库函数 `cv::cvtColor` 进行颜色空间转换。使用公式 $Y = 0.299R + 0.587G + 0.114B$ 进行灰度化。

```
1 cv::Mat toGrayscale(const cv::Mat& image) {
2     if (image.channels() == 1) {
3         return image.clone();
4     }
5     if (image.channels() != 3 || image.depth() != CV_8U) {
6         throw std::runtime_error("仅支持 8 位三通道彩色图像转换为灰度");
7     }
8
9     cv::Mat gray;
10    cv::cvtColor(image, gray, cv::COLOR_RGB2GRAY);
11    return gray;
12 }
```

3.4 图像缩放

在 [ImageOps.cpp](#) 中实现。调用 opencv 库函数 `cv::resize` 进行图像缩放。默认使用双线性插值。

```
1 cv::Mat scaleByPercentage(const cv::Mat& image, double scale, int interpolation) {
2     if (image.empty()) {
3         throw std::runtime_error("无法缩放空图像");
4     }
5     if (scale <= 0.0) {
6         throw std::runtime_error("缩放比例必须大于 0");
7     }
8     if (scale == 1.0) {
9         return image.clone();
10    }
11
12    cv::Mat result;
13    cv::resize(image, result, cv::Size(), scale, scale, interpolation);
14    return result;
15 }
```

3.5 图像读写

在 [ImageLoader.cpp](#) 中实现。对于 P2, P3 格式直接输出 ASCII 码，对于 P6 格式需求二进制输出。

```
1 void writeAscii(const cv::Mat& image, std::ostream& os, int /*maxValue*/, bool
   isColor) {
2     // write ASCII format PPM/PGM image
3     const int width = image.cols;
4     const int height = image.rows;
5
6     if (isColor) {
7         for (int row = 0; row < height; ++row) {
8             for (int col = 0; col < width; ++col) {
```

```

9         const auto pixel = image.at<cv::Vec3b>(row, col);
10        os << static_cast<int>(pixel[0]) << ' '
11           << static_cast<int>(pixel[1]) << ' '
12           << static_cast<int>(pixel[2]) << '\n';
13    }
14 }
15 } else {
16     for (int row = 0; row < height; ++row) {
17         for (int col = 0; col < width; ++col) {
18             os << static_cast<int>(image.at<std::uint8_t>(row, col)) << ' ';
19         }
20         os << '\n';
21     }
22 }
23 }

```

```

1 void writeBinaryP6(const cv::Mat& image, std::ostream& os) {
2     // write binary P6 format PPM image
3     if (image.type() != CV_8UC3) {
4         throw std::runtime_error("二进制 P6 输出仅支持 8 位 3 通道图像");
5     }
6     const std::size_t total = static_cast<std::size_t>(image.total()) *
image.elemSize();
7     os.write(reinterpret_cast<const char*>(image.data), static_cast<std::streamsize>
(total));
8 }

```

3.6 图像压缩

在 [ImageLoader.cpp](#) 中实现。

首先为了利用图像局部相关性，将所有像素对其左侧数字做差分，而对于不同通道独立处理。实现函数

`std::vector<std::uint8_t> buildResidualChannel(const cv::Mat& image, int channel)` 和 `std::vector<std::uint8_t> reconstruct(const std::vector<std::uint8_t>& residuals, int width, int height)`。

考虑到差分后数字出现频率差距悬殊，使用 Huffman 编码进行无损压缩。

首先统计各种数字出现频率，实现 `std::array<std::uint64_t, 256> buildHistogram(const std::vector<std::uint8_t>& data)`。然后对其建立 Huffman 树。为了方便编码，考虑求出规范哈夫曼码字长度，实现 `std::array<int, 256> buildCodeLengths(const std::array<std::uint64_t, 256>& histogram)`。最后根据码字长度生成具体编码，实现 `HuffmanTable buildCanonicalTable(const std::array<std::uint8_t, 256>& lengths)`。

读写只需要按照指定的文件规范操作即可：

```
1  /*
2  * Compression format:
3  * [magic "HFM" (3 bytes)]
4  * [width (4 bytes)]
5  * [height (4 bytes)]
6  * [maxValue (2 bytes)]
7  * [channels (1 byte)]
8  * [Huffman tables (256 bytes each channel)]
9  * [dataBitCount (4 bytes)] [encoded data (variable)]
10 */
```

4. 程序运行方式

编译程序：

```
1  cmake -S . -B build
2  cmake --build build
```

可对程序使用 `--help` 指令获取使用说明。