

浙江大学

计算机视觉(本科)实验报告

作业名称: HW1 制作个人视频

姓 名: 夏子渊

学 号: 3230103043

电子邮箱: RukawaYuan1110@gmail.com

学 院: 计算机科学与技术学院

专 业: 软件工程

指导教师: 宋明黎/潘纲

报告日期: 2025 年 04 月 25 日

Table of Contents

1 功能简述及运行说明	4
1.1 功能简述	4
1.2 项目结构	4
1.3 运行说明	4
2 开发与运行环境	5
2.1 操作系统	5
2.2 安装依赖	5
3 算法的基本思路、原理、及流程	5
3.1 基本思路	5
3.2 数学原理	6
3.2.1 图像缩放	6
3.2.2 交叉淡入淡出过渡效果	6
3.2.3 视频帧率与时序控制	6
3.3 算法流程	6
3.3.1 初始化阶段	6
3.3.2 视频生成阶段	6
3.3.3 资源释放阶段	6
4 具体实现——关键代码、函数与算法	7
4.1 文件处理模块	7
4.1.1 目录扫描与文件分类	7
4.1.2 文件数量验证	7
4.2 图像处理模块	8
4.2.1 图像加载与尺寸归一化	8
4.3 视频合成模块	8
4.3.1 标题帧生成	8
4.3.2 交叉淡入淡出效果	9
4.3.3 幻灯片播放控制	9
4.3.4 原始视频处理	10
4.4 文字叠加模块	10
4.4.1 居中标题文字添加	10
4.4.2 底部字幕文字添加	11
5 实验结果与分析	11
5.1 实验结果	11
5.2 结果分析	13
6 结论与心得体会	14
6.1 实验结论	14

6.2 心得体会	14
6.3 实验收获	14
7 参考文献	14

1 功能简述及运行说明

1.1 功能简述

本实验对输入的一个彩色视频与五张以上照片，用 OpenCV 实现以下功能或要求：

1. 命令行格式: `./video_maker <input_path>`, 例如 `./video_maker ../input`, 其中 `input` 文件夹下有一个 avi 格式的视频与至少 5 张 png/jpg/jpeg 格式的图片, 以及对应的输出视频 `output_video.avi`
2. 将输入的视频与照片处理成同样长宽 (本实验中定为 720×540) 后, 合在一起生成一个视频
3. 在新视频中, 编程生成一个片头, 然后按幻灯片形式播放这些输入照片, 最后按视频原来速度播放输入的视频
4. 新视频中要在底部打上含自己学号与姓名信息的字幕: `Student ID: 323`
`0103043 Name: Xia Ziyuan`
5. 编程实现镜头切换效果: 在两张幻灯片之间增加平滑过渡的效果, 即一张幻灯片淡出, 另一张幻灯片淡入

1.2 项目结构

本项目的结构如下:

```
.  
└── video_maker.cpp      # main program  
└── CMakeLists.txt      # CMake configuration  
└── README.md            # this file  
└── report.pdf          # report  
└── build                # build directory  
└── input                # example input images/video and output video  
    └── slide_1.png  
    └── slide_2.png  
    └── slide_3.png  
    └── slide_4.png  
    └── slide_5.png  
    └── input_video.avi  
    └── output_video.avi
```

1.3 运行说明

- 构建: 使用 clangd 作为编译器, CMake 进行构建, 在 build 目录下得到可执行的文件。在提交的压缩包中已经上传了 build 文件夹, 其中可执行文件位于 `build/video_maker` 目录下。如果想要自行构建, 请先切换到根目录下, 然后在终端中运行以下命令:

```
mkdir build  
cd build  
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 ..  
make -j
```

即可得到对应的可执行文件。

- 运行：我在 `./input` 目录下提供了一个示例，包含一个 avi 格式的视频与 5 张照片，以及运行之后对应的输出视频。如果想要在这个示例上运行代码，可以切换至 build 目录下并执行以下命令：

```
./video_maker ../input
```

或者也可以提供自己的输入，放在 input 目录下，然后切换至 build 目录并运行相同的命令，在 input 目录下得到 output_video.avi 格式的输出视频。

2 开发与运行环境

2.1 操作系统

在 MacOS (M4 Chip) 上进行，硬件配置为 10 核 CPU，24GB 内存

2.2 安装依赖

实验使用 C++ 作为编程语言，因为需要 `<opencv2/opencv.hpp>` 头文件，因此需要先安装 opencv 库。同时因为构建时需要使用 CMake，因此也需要安装 Cmake。使用 brew 包管理器进行安装，具体命令如下：

```
brew install cmake  
brew install opencv
```

3 算法的基本思路、原理、及流程

3.1 基本思路

本算法实现了一个自动化的视频生成系统，能够将指定文件夹中的图片和视频素材合成一个具有专业效果的视频。视频包含三个主要部分：

1. 标题页展示
2. 图片幻灯片播放（带过渡效果）
3. 原始视频播放

3.2 数学原理

3.2.1 图像缩放

在图像缩放过程中，使用双线性插值算法来保持图像质量。双线性插值的数学表达式为：

$$f(x, y) = \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right)$$

其中 Q_{ij} 是源图像中最近的四个像素点。

3.2.2 交叉淡入淡出过渡效果

交叉淡入淡出效果使用加权平均算法实现，数学表达式为：

$$D_{\text{out}} = (1 - \alpha) \times S_1 + \alpha \times S_2$$

其中：

- S_1 是第一幅图像的像素值
- S_2 是第二幅图像的像素值
- α 是过渡系数，从 0 到 1 线性变化
- D_{out} 是输出图像的像素值

3.2.3 视频帧率与时序控制

视频时序控制基于帧率(FPS)和持续时间计算帧数：

$$\text{帧数} = \text{FPS} \times \text{持续时间(秒)}$$

3.3 算法流程

3.3.1 初始化阶段

1. 解析命令行参数获取文件夹路径
2. 扫描文件夹获取图片和视频文件
3. 验证文件数量是否符合要求

3.3.2 视频生成阶段

1. 创建标题帧并写入视频
2. 加载所有图片并调整尺寸
3. 实现图片幻灯片播放：
 - 每张图片显示 80% 的指定时间
 - 图片间使用 20% 的时间做过渡效果
4. 加载并播放原始视频

3.3.3 资源释放阶段

1. 关闭视频读写器
2. 释放内存资源

4 具体实现——关键代码、函数与算法

4.1 文件处理模块

4.1.1 目录扫描与文件分类

1. 使用 `boost::filesystem` 遍历输入目录
2. 对每个文件提取扩展名并转为小写
3. 根据扩展名分类存储:
 - .jpg/.jpeg/.png 存入 `imagePaths` 向量
 - .avi/.mp4 存入 `videoPath` 变量

代码实现如下:

```
for (const auto& entry : fs::directory_iterator(folderPath)) {  
    string path = entry.path().string();  
    string ext = entry.path().extension().string();  
    transform(ext.begin(), ext.end(), ext.begin(), ::tolower);  
  
    if (ext == ".jpg" || ext == ".jpeg" || ext == ".png") {  
        imagePaths.push_back(path);  
    } else if (ext == ".avi" || ext == ".mp4") {  
        videoPath = path;  
    }  
}
```

4.1.2 文件数量验证

验证逻辑:

- 至少需要 5 张图片和 1 个视频文件
- 不满足条件时立即返回错误

实现代码:

```
if (imagePaths.size() < 5) {  
    cerr << "Need at least 5 images in the folder" << endl;  
    return -1;  
}  
if (videoPath.empty()) {  
    cerr << "No video file found in the folder" << endl;  
    return -1;  
}
```

4.2 图像处理模块

4.2.1 图像加载与尺寸归一化

处理流程:

- 使用 `imread` 加载图像
- 检查图像是否有效
- 使用 `resize` 统一调整为 `OUTPUT_SIZE`
- 存入 `images` 向量

核心代码:

```
vector<Mat> images;
for (const auto& path : imagePaths) {
    Mat img = imread(path);
    if (img.empty()) {
        cerr << "Could not read image: " << path << endl;
        continue;
    }

    Mat resizedImg;
    resize(img, resizedImg, OUTPUT_SIZE);
    images.push_back(resizedImg);
}
```

4.3 视频合成模块

4.3.1 标题帧生成

实现流程:

1. 创建深棕色背景画布
2. 添加居中标题文字
3. 添加底部学生信息
4. 写入指定帧数（3秒）

代码实现如下:

```
Mat titleFrame(OUTPUT_SIZE, CV_8UC3, Scalar(50, 50, 150));
addTextCentered(titleFrame, "Xia Ziyuan's Generated Video", 1.4);
addBottomText(titleFrame, STUDENT_INFO);

for (int i = 0; i < TITLE_DURATION * FPS; ++i) {
```

```
    writer.write(titleFrame);
}
```

4.3.2 交叉淡入淡出效果

使用加权平均公式: $D_{out} = (1 - \alpha) \times S_1 + \alpha \times S_2$, 通过设置 α 从 0 逐渐过渡到 1, 实现从一帧逐渐过渡到另一帧的效果, 实现代码如下:

```
void crossFade(const Mat& src1, const Mat& src2, Mat& dst, double alpha) {
    addWeighted(src1, 1.0 - alpha, src2, alpha, 0.0, dst);
}
```

4.3.3 幻灯片播放控制

在播放时间的分配上, 设置正常播放和过渡的时间占比如下: 单张图片显示 80% 时间 (2.4 秒, 过渡效果 20% 时间 (0.6 秒), 实现代码如下:

```
for (size_t i = 0; i < images.size(); ++i) {
    // 显示当前图片
    for (int j = 0; j < SLIDE_DURATION * FPS * 0.8; ++j) {
        Mat frame = images[i].clone();
        addBottomText(frame, STUDENT_INFO);
        writer.write(frame);
    }

    // 过渡效果
    if (i < images.size() - 1) {
        for (int j = 0; j < SLIDE_DURATION * FPS * 0.2; ++j) {
            double alpha = static_cast<double>(j) / (SLIDE_DURATION
* FPS * 0.2);
            Mat frame;
            crossFade(images[i], images[i+1], frame, alpha);
            addBottomText(frame, STUDENT_INFO);
            writer.write(frame);
        }
    }
}
```

4.3.4 原始视频处理

在合成视频的末尾添加原视频：打开视频文件 >> 逐帧读取 >> 调整尺寸 >> 添加文字 >> 写入输出，实现代码如下：

```
VideoCapture cap(videoPath);
Mat frame;
while (cap.read(frame)) {
    Mat resizedFrame;
    resize(frame, resizedFrame, OUTPUT_SIZE);
    addBottomText(resizedFrame, STUDENT_INFO);
    writer.write(resizedFrame);
}
cap.release();
```

4.4 文字叠加模块

4.4.1 居中标题文字添加

函数的实现上，首先计算文本尺寸和基线，再计算居中位置，最后绘制文字。关键技术点如下：

1. 精确的文本居中计算：

- 水平居中：(width - textWidth)/2
- 垂直居中：(height + textHeight)/2 - baseline/2（考虑字母下沉部分）

2. 文字阴影效果：

- 先绘制黑色阴影（偏移 2 像素）
- 再绘制主文字，增强视觉层次感

3. 抗锯齿处理：

- 通过 putText 的线宽参数实现文字边缘平滑

```
void addTextCentered(Mat& frame, const string& text, double
fontScale = 1.0, Scalar color = Scalar(255, 255, 255)) {
    int baseline = 0;
    Size textSize = getTextSize(text, FONT_HERSHEY_SIMPLEX,
fontScale, 2, &baseline);
    Point textOrg((frame.cols - textSize.width) / 2, (frame.rows +
textSize.height) / 2);
    putText(frame, text, textOrg, FONT_HERSHEY_SIMPLEX, fontScale,
color, 2);
}
```

4.4.2 底部字幕文字添加

与“居中标题文字添加”模块类似，底部字幕文字添加部分在视频帧底部添加字幕，具体实现如下：

- 固定位置：距离底部 10 像素
- 固定字体大小：0.9
- 白色文字

```
void addBottomText(Mat& frame, const string& text) {  
    int baseline = 0;  
    Size textSize = getTextSize(text, FONT_HERSHEY_SIMPLEX, 0.9, 2,  
    &baseline);  
    Point textOrg((frame.cols - textSize.width) / 2, frame.rows -  
    10);  
    putText(frame, text, textOrg, FONT_HERSHEY_SIMPLEX, 0.9,  
    Scalar(255, 255, 255), 2);  
}
```

5 实验结果与分析

在压缩包的 input 目录下已有提供示例输入与输出，以下分析以此为例：

5.1 实验结果

1. 封面



按照实验要求生成了封面，如上图所示。封面的背景整体为深棕色，中间为视频标题 **Xia Ziyuan's Generated Video**，下部为要求的字幕 **Student ID: 3230103043 Name: Xia Ziyuan**，整体播放时间为 3 秒 72 帧，符合要求。

2. 幻灯片



Student ID: 3230103043 Name: Xia Ziyuan

如图所示，依次播放输入文件夹中的 5 张图片，并在底部添加字幕，帧率为 24FPS。图片以幻灯片形式播放，每张幻灯片时间为 3 秒，两张幻灯片之间有 0.6 秒用于渐变过渡，效果如下：



Student ID: 3230103043 Name: Xia Ziyuan

3. 原视频



按照要求以原视频帧率播放原视频内容，并在底部添加了包含学生信息的字幕。

5.2 结果分析

从多个角度进行分析：

1. 功能完整性：

- 系统实现了标题生成、幻灯片播放、视频处理全部要求功能
- 各项参数精确达到设计要求

2. 视觉效果：

- 过渡平滑自然，无可见瑕疵
- 文字显示清晰，布局合理
- 图像缩放质量良好

3. 性能表现：

- 实时处理能力达标（稳定 24fps）
- 资源消耗在合理范围内
- 处理速度与视频长度线性相关

4. 鲁棒性：

- 能正确处理不同分辨率的输入
- 对异常输入有完善错误处理
- 输出视频规格严格一致

6 结论与心得体会

6.1 实验结论

通过本次实验，我成功实现了一个基于 OpenCV 的视频与照片处理程序。该程序能够读取指定文件夹中的一个视频文件和若干照片，将它们处理成统一的分辨率，并生成一个新的视频。新视频包含一个片头、幻灯片形式的照片展示以及原始视频内容，并在底部添加了包含学号和姓名信息的字幕。此外，我还实现了平滑的镜头切换效果，使视频过渡更加自然。

6.2 心得体会

在实验过程中，我遇到了不少困难和报错，但经过查询相关资料与调试最终得以解决：

- 在读取文件夹内容时，最初遇到了文件路径解析的问题。由于不同操作系统对路径的处理方式不同，导致在某些系统上路径解析失败。通过使用 boost::filesystem 库处理文件路径，解决了这一问题。
- 另一个问题是在视频帧率的同步。在将照片和视频合并时，需要确保它们的帧率一致。通过设置统一的帧率（24fps），并调整照片的持续时间，最终实现了视频的流畅播放。
- 在将视频和照片调整为统一分辨率时，最初尝试直接使用 resize 函数，但发现某些图像在调整后会出现失真。通过调整 resize 函数的插值方法，最终解决了这一问题。
- 实现平滑的镜头切换效果（cross fade）时，直接使用 addWeighted 函数发现过渡效果不够平滑。通过自己设计了一个函数并调整 alpha 参数的计算方式，最终实现了平滑的过渡效果。

6.3 实验收获

通过本次实验，我深入学习了 OpenCV 的图像和视频处理功能，包括图像读取、调整大小、添加文字、视频帧处理等。在解决文件路径处理、图像和视频帧率同步等问题时，也学会了如何使用 boost::filesystem 库和 OpenCV 的高级功能，提高了系统编程的能力。

7 参考文献

- OpenCV 官方网站：<https://opencv.org>
- OpenCV 官方文档：<https://docs.opencv.org/4.x/d1/dfb/intro.html>