isep
École d'ingénieurs du numérique

Development of multimedia app

Project Report

25 May 2025

Students Teiva TRILLARD

Anthony NORA

Morgane LE BON

Yohan BONNET

Éric TONG

Table of contents

I - Summary of the semester

For the second half of our A2 year in ISEP, we have had the opportunity to attend the Multimedia development course. In this course, we learned what multimedia was, which is a set of techniques and products that permit the simultaneous and interactive usage of different ways to represent information (text, video, images, sounds, etc..). We also learned how to use the C++ language with object-oriented principles and especially how to work with the OpenCV library. During our first TD, we had to develop different short functions to understand how to compile and run with libraries and shared libraries. We used a Makefile to launch those programs. That is useful to store and execute multiple lines of shell code in one time.

As the course progressed, we deepened our understanding of memory management in C++. We explored key concepts such as passing by value versus passing by reference, and how these affect function behaviour and memory usage. We also discussed variable scope, making the distinction between local and global variables, as well as the differences between stack and heap memory. These topics were reinforced using pointers and arrays, helping us to better understand how memory is allocated and accessed. Additionally, we learned to identify common issues due to incorrect memory handling.

Moving forward, we studied structure and object-oriented programming. We used typedef to simplify type declarations and learned how to define and use classes effectively. Important object-oriented principles such as encapsulation, inheritance, and polymorphism were introduced. We practiced using access modifiers to control visibility and created constructors and destructors to manage object lifecycles. These concepts allowed us to design more modular and reusable code, which is essential for our project.

A significant part of the course focused on the OpenCV library, a powerful tool for multimedia development. OpenCV is an open-source computer vision library that provides tools for processing and analysing images and videos in real time.

We learned how to manipulate images and videos using various functions. This included adjusting brightness and contrast, resizing and rotating media, applying filters and blurs, and saving videos recorded from a camera. We also explored drawing on images by creating shapes and handling mouse interactions to add interactivity to our application. These exercises gave us practical experience useful to build dynamic and responsive programs for the final project.

II - Description of the project

In our application, we leveraged a wide range of OpenCV functions to implement various image processing techniques, all encapsulated within a modular object-oriented architecture. For basic image management, we used **cv::imread** and **cv::imshow** to load and display images, while **cv::waitKey** and **cv::destroyWindow** managed user interaction and window lifecycle. The image class encapsulates the current image and its history, providing methods for undo and version restoration, ensuring a robust workflow for users.

For morphological operations, such as erosion and dilation, we used **cv::getStructuringElement**, **cv::erode**, and **cv::dilate**. These are implemented in the erosion class, which provides both grayscale and color variants. The grayscale methods convert images using **cv::cvtColor** and apply thresholding with **cv::threshold**, while color methods operate directly on the color channels. Trackbars created with **cv::createTrackbar** allow users to interactively adjust parameters.

Edge detection is handled by the canny class, which uses **cv::Canny** for edge extraction. The class also provides interactive threshold adjustment via trackbars, and ensures grayscale conversion for consistent results. For resizing, the resize class uses **cv::resize**, supporting both scaling factors and explicit dimensions, with user input guiding the process.

Brightness adjustment is managed by the brightness class, which manipulates pixel values directly and displays results for user confirmation. Panorama stitching is implemented in the panorama class, utilizing OpenCV's **cv::Stitcher** to combine multiple images into a seamless panorama.

Background removal is addressed in the backgroundRemover class, which offers three techniques: foreground extraction with **cv::grabCut**, thresholding with **cv::threshold**, and chroma keying using **cv::cvtColor**, **cv::inRange**, and **cv::bitwise_not**. User interaction is facilitated by mouse callbacks (**cv::setMouseCallback**) for color selection.

Face detection is encapsulated in the faceDetection class, which loads a Haar cascade with **cv::CascadeClassifier::load** and detects faces using detectMultiScale. Detected faces are highlighted with **cv::rectangle**, and the class supports both static images and live webcam input via **cv::VideoCapture**.

We can adjust brightness and darken the picture using the **cv::Scalar** class which can add or remove a scalar value to all pixels of the picture, effectively brightening or darkening the picture, no matter whether it's greyscaled or not.

All these functionalities are orchestrated through the interface class, which presents a menu-driven user interface and delegates operations to the appropriate classes. Each processing technique is encapsulated in its own class, following OOP principles, promoting code reuse, maintainability, and scalability. The application's design ensures that each image processing operation is modular, with clear separation of concerns, making it easy to extend or modify individual features without impacting the overall system.

This project all comes together behind a Qt6 graphical interface. That way users can easily load and modify their pictures without having to know the exact commands and makes the opencv functionalities very easy to use. This user interface communicates with the interface class to transmit the actions of the user and perform them onto the image itself.

III - Planned features

Firstly, we planned on adding the ability to draw on the picture using **cv::setMouseCallback** to handle mouse events, and implement drawing operations (line, squares, circles) using **cv::line**, **cv::circle** and **cv::polylines**. The user could then draw directly on the picture and it would be updated. We could use the same mouse event handling function in order to select an area using a lasso. The selected area would be managed as a simple binary **cv::Mat** and only allow modifications on the image when it matches the selected area.

Secondly, we also could have added real time filters to this project. We would have used filters such as **cv::blur**, **cv::GaussianBlur**, **cv::Laplacian**, or **cv::Sobel**, maybe repeatedly to intensify the effect using a slider.

Finally, using the latest opencv technology we could add AI-powered features via the DNN module of opencv. That way we could do operations such as style transfer, object detection, or segmentation using AI models that are trained just for that.

IV - Talk about the project itself and how you handled it

A) Difficulties and solutions

When we started working on the project, git would for some reason not allow other members to contribute to the project apart from two members who were still on the git project. We did not want to take too long debugging our git so instead we gave our parts via mail to a member that had access to the git, and he commit our parts for us in the git.

During the project's development, there would be times when the windows would not close, but it was quickly fixed by moving the instructions in order that would not freeze the window when closing.

B) Teamwork

We split the work among the 5 members of the team such that each person did a feature of the project, and the members with more experience in C++ helped the ones with less experience, so everyone worked at the same speed more or less.

C) Tools

We used Visual Studio, and CLion to code the project, the OpenCV library, and the Qt6 library for the user interface. We shared the project together using Git and communicated through WhatsApp and Teams.

Conclusion

This project allows us to put into practice all the skills we acquired during the semester, such as object-oriented programming in C++ and the use of the OpenCV library for image processing.

Object-oriented programming enabled us to structure our code in a modular way, by separating each feature into a dedicated class. This approach improves code readability, reusability, and facilitates future development of project.

The OpenCV library allowed us to implement various features such as face detection, background removal, brightness and contrast adjustments, image rotation and resizing, interactive drawing on images, panorama creation, and the application of filters (like blurring and edge detection). The use of trackbars and mouse events also helped make the application interactive and intuitive.

Finally, this group project taught us how to organize ourselves, divide tasks among team members, and structure our code so that everyone could contribute easily, even with different levels of experience in C++.

In conclusion, this work allowed us not only to improve our technical skills, but also to better understand the challenges of multimedia development and the importance of collaboration in a software project.