HW1 보고서 한용옥 2020203090

과제 목표, 전체 구조

과제 설계의 핵심은 아래 두가지이다
원본 프레임은 항상 보존
마우스 버튼을 놓는 순간에만 주석(사각형) 확정해서 저장
구현은 크게 두 클래스로 나누었다
`fpstimer` : 목표 FPS에 맞게 재생 속도를 조절하는 타이머(경과 측정 + 남는 시간 대기)
`video_player` : 비디오 로드, 프레임 로드, 재생/일시정지, 키/마우스 입력, 사각형 주석 저장/표시 등 메인 로직 담당

클래스 설계 이유

`fpstimer`
렌더링/입력 로직과 시간 제어를 분리하려 별도 클래스로 제작 `getTickCount()`로 한 프레임에 걸린 시간을 재고, 남은 시간만큼 `sleep_for`로 쉬게 해서 목표 FPS를 맞춤 재생 로직을 건드리지 않고도 FPS만 바꾸기 편하다

`video_player`
플레이어가 해야 할 일을 한 곳에 모음 생성자에서 파일 열고(웹캠 X), 윈도우 만들고, 마우스 콜백 등록한 뒤 루프를 돈다 주석은 `unordered_map<int, Mat> annot`에 저장한다(키: 프레임 번호, 값: 주석이 반영된 프레임 이미지) 드래그 중에는 화면에만 미리 보이고, 버튼을 놓을 때 확정해서 맵에 넣는다 따라서 수정한 프레임만 메모리를 쓰고, 원본은 깨끗하게 유지된다

주요 메서드 상세 — 구현 의도와 OpenCV 적용

`fpstimer::start / end / until_fps_delay`

의도 렌더링·입력 로직과 FPS 맞추는 시간 제어를 분리, 재생 품질을 일정하게 만들고 디버깅도 쉽게 하려는 목적
구현 · OpenCV 사용
`start()`에서 `cv::getTickCount()`로 시작 시점을 저장
`end()`에서 `(getTickCount() - start_T) / getTickFrequency()`로 한 프레임이 실제로 걸린 초 단위 경과 시간을 얻음
`until_fps_delay()`는 `목표_프레임시가ㄴ (1/fps) - 실제_경과`를 계산해서 부족분만
`std::this_thread::sleep_for`로 잠깐 멈춤
`waitKey()`로 시간을 때우면 GUI 이벤트 처리와 결합돼서 제어가 꼬일 수 있어서, 시간 지연은 C++ 표준 라이브러리로, 키 입력은 OpenCV로 분리함

`video_player::loop`

의도 입력 → 상태 갱신 → 렌더 → 시간조절의 게임 루프 패턴을 단순하게 유지
구현 · OpenCV 사용
루프 초입에 `fpstimer.start()`
`cv::pollKey()`로 논블로킹(non-blocking) 키 입력을 받음 스페이스(재생/일시정지), N/P(프레임 스텝), Q/Esc(종료) 같은 핫키를 분기
`playing`이면 `step(+1)` 호출로 다음 프레임으로 이동
매 프레임 `display()`로 화면 갱신
루프 말미에 `fpstimer.end()`로 경과를 재고 `until_fps_delay()`로 FPS 맞춤

`video_player::display`

의도 원본을 절대 손대지 않고, 표시용 복사본(canvas) 위에서만 덧그려서 안정성 확보
구현 · OpenCV 사용

현재 프레임 번호 `i`에 대해:
`annot.count(i)`가 있으면 `annot[i].clone()`을, 없으면 `base.clone()`을 표시용 캔버스로 사용
드래그 중이면 `cv::rectangle(canvas, norm_rect(start_pt, cur_pt), …)`로 임시 사각형을 오버레이
이 단계에서는 저장하지 않음 → 미리보기 전용
마지막에 `cv::imshow(window, canvas)`로 화면 출력
이렇게 하면 실수로 원본을 덮어쓸 일이 없고, 미리보기(드래그 중)와 확정(드롭 순간)이 자연스럽게 분리됨

`video_player::on_mouse`

의도 누르는 순간 시작점 기록, 움직일 때 미리보기, 놓는 순간 확정
구현 · OpenCV 사용
생성자에서 `cv::setMouseCallback(window, &video_player::on_mouse, this)`로 콜백 등록
`EVENT_LBUTTONDOWN`: `playing = false`로 일시정지(편집 모드), `start_pt = cur_pt = 현재 좌표` 곧바로 `display()` 호출해서 시각 피드백 빠르게 제공
`EVENT_MOUSEMOVE`: `dragging` 상태면 `cur_pt` 갱신 후 `display()`로 실시간 프리뷰
`EVENT_LBUTTONUP`: `dragging = false`, `commit_rect()` 호출로 확정 저장 확정 후 `display()`로 결과 반영

`video_player::commit_rect`

의도 확정 시에만 저장해서 메모리 낭비 방지 + 원본 불변성 유지
구현 · OpenCV 사용
현재 프레임 `i`가 `annot`에 없다면, 처음 편집하는 프레임이므로 `annot[i] = base.clone()`으로 주석본 저장소를 생성
그 위에 `cv::rectangle(annot[i], norm_rect(start_pt, cur_pt), …)`로 최종 사각형을 그려서 프레임 주석본 완료
항상 `annot[i]`에만 그린다. `base`에는 절대 그리지 않음 → 원본 보존 보장

`video_player::step(int delta)`

의도 프레임 인덱스 이동을 한 곳에서 처리하고, 경계(0 ~ total-1) 를 안전하게 보정
구현 · OpenCV 사용
새 인덱스 `next = clamp(cur_idx + delta, 0, total-1)`.
`next != cur_idx`일 때만 `load_frame(next)`로 실제 로드 → 불필요한 접근 방지
스텝키(N/P)를 눌렀을 때 자동으로 `playing = false`로 만들어서 정밀 편집 모드에 맞춰줌

`video_player::load_frame(int i)`

의도 임의 프레임 접근 시에도 읽기 실패 시 화면 유지로 끊김 최소화
구현 · OpenCV 사용
`cap.set(cv::CAP_PROP_POS_FRAMES, i)`로 프레임 단위 접근 요청
그 다음 `cap.read(tmp)`로 디코딩. 성공하면 `base = std::move(tmp)`, `cur_idx = i`
실패하면 아무것도 바꾸지 않아 현재 화면 유지