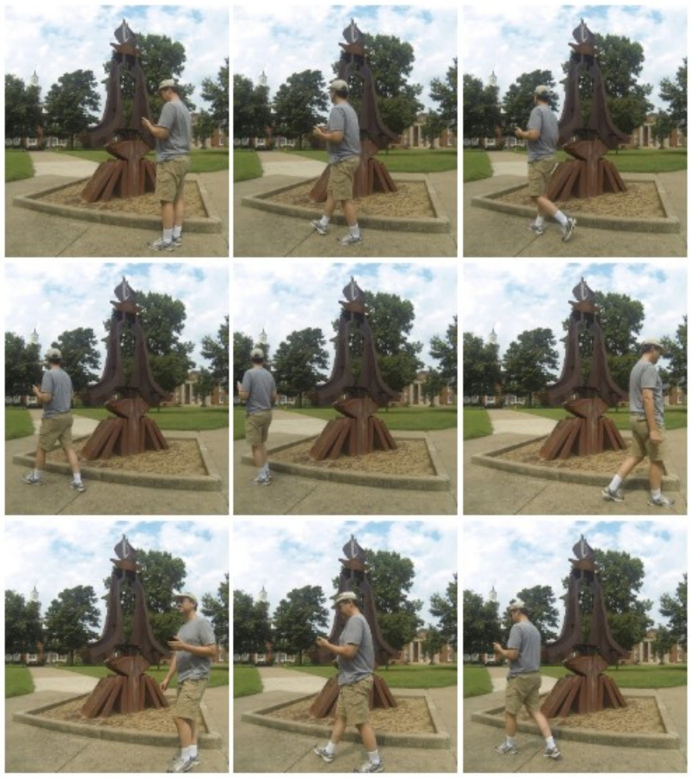
Team project - median filter to remove noise from picture

Submit Assignment

Due May 18 by 7pm **Points** 100 **Submitting** a file upload **File Types** cpp, h, and pdf **Available** until May 18 at 7pm

Suppose we want a photograph of this sculpture, but there was a person walking around there at the same time. Somehow he managed to be in every single frame! We want a single photo of the sculpture, without anyone in the picture.



Notice the guy had been moving a lot. You can see that the most of sculpture is visible in each image. He only blocked part of the sculpture, and since he was in a different position in each photo, he usually blocked a different part of the sculpture each time. If we can determine which pixel colors are part of the sculpture and the background (what we want), and which pixel colors are part of the person (what we don't want), we could create a photo without him in it.

It turns out there is a way to do this. We can use an image processing technique called the median filter. This is a method for removing noise from a set of similar images.

In the median filter, we compare individual pixels from each image. We start out by making the assumption that most of the pixels in each image at a specific coordinate (x, y) are same color or very close in value. These are the colors we want. Sometimes, at a particular pixel in a particular image, that pixel will be part of the something else and that pixel will have a very different color from the other pixels at the same coordinate in the other images. These different colored values are the noise we are looking for and trying to remove.

In order to pick the correct color for the pixel at each (x, y) coordinate, we can read the color value for that particular coordinate in all the images at the same time, placing them into an array or other container. We can then sort those values, and pick the middle value, i.e., the median value for that list of numbers. Once we know that median value, we can write it out to our filtered image. Of course we have to remember that each pixel really contains three values: one for red, one for green, and one for blue. That means for each pixel, we have to find three median values.

If we do the above process for each pixel, and if we do it for the red, green, and blue value in each pixel, then the final result will create a good photo without that interloper in it.

Your Task:
You are given some **PPM image files**. The images are all the same height and width. You will open all the images at the same time for reading, and then open a file result.ppm for writing. Create the proper PPM header. Read the header values from the first file to do that. Make sure to skip the header in the other files.

Once you are ready to read the pixel color values, read all the values in the files, one integer at a time. The first value in the first file, the first value in the second file, the first value in the third file, and so on up to the first value in the last file. Calculate the median, and write the result to the result image. Continue on to the second value and follow the same process, and then the remaining values the same way. Your result image should be the image without the distraction.

An interesting improvement on the idea of using an array is to use a min-heap and a max-heap to collect the pixel integers. If we keep those almost the same size, then we can find the median without sorting. *This is required; do NOT simply sort the containers to get the median.*

Rubric - Median Filter

Criteria	Ratings		Pts
Specification Description of what the program does	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
Analysis An analysis of the specification that identifies details about the inputs, the overall process, and the outputs of the program from the users point of view.	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
Design A list of all source code files with a short description of each one.	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
Implementation A separate page for each source file (.h and .cpp). Each one must be clearly visible. Do not put more than about 25-30 lines per page. If your function is longer, split it up over several pages. Put comments alongside the code explaining how it works. These comments cannot be just C++ comments, but must be annotations added with Latex or other improved formatting methods.	**60.0 pts Full Marks**	**0.0 pts No Marks**	60.0 pts
Test Show screenshots of your program as it is being executed. Use Catch2 with a separate SECTION for each function test. Describe what is being tested by each one.	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
Documentation Make sure no code or comments are too small to see easily. Use proper margins so the code is all visible on the page with logical breaks using \newpage if needed to keep the code and comments coherent. Use correct spelling and grammar. Don't just paraphrase the code, but add comments which help the reader understand the purpose of the code and how it works. Use different fonts and colors where appropriate to enhance the quality. Use diagrams to help illustrate the algorithms.	**10.0 pts Full Marks**	**0.0 pts No Marks**	10.0 pts
			Total Points: 100.0