Group: Benjamin Davidson and Eric Daniel Tauscher

Course: CMSC388Z

Visuadi

**Summary of proposal and goals**:

**Project description**:

The purpose of visuadi is to create an interesting way of generating digital sounds through pictures (.jpg is only being supported at the moment). People who may benefit from this program are music producers or individuals that create foley for films or other types of video content.

**Usage instructions**:

To see if the program will compile, run the command cargo build. There are no operating specific libraries that should hinder anyone to run this program (has only been tested on Windows and Linux). To run the program, run the command:

cargo run -- --pic=<pic> --name=<name> --pts=<pts>

In your own implementation, <pic> is replaced with the desired .jpg file to processes the audio. The <name> field is simply the name of the result audio file (it is advised to end this file with a .wav extension).

The <pts> is an optional argument for the image to audio data process that represents the number of points in the moving average filter. Although setting the number of points for the moving average filter is optional, it is highly recommended for most circumstances that this field be set to a value between 2 and 16. Visuaudi can output interesting sounds, but depending on what picture is being used, the high end audio frequency content can be very harsh and loud.

When using this program, it is worth noting a few things: First, the volume on your computer should be turned down to a reasonable volume (the audio is normalized by default). Additionally, considering what pictures to use based on their *size* and not just their appearance is also a good method to experiment with (since smaller pictures will result in shorter, higher pitched volumes).

**Challenges and surprises**:

To our surprise, there were not many challenges during the creation of this project. We were able to implement basic image to audio data conversion very quickly, without any strange bugs in the making. However, implementing the moving average filter was surprisingly challenging.

Although the concept of a moving average filter is easy to understand, implementing it in an efficient way provided some challenges. First, we tried to do it recursively; this approach would have definitely resulted in much cleaner code, but deadlines were approaching, and we needed results. The way it is currently implemented runs in linear time, without having to do to many extra loops (which is the benefit of doing it recursively).

**Observations about Rust**: