Maarten ‘t Hart – 31 december 2019

# Audio Eye

This document is a description of a concept that may allow blind people to use their ears to “see”. This would be achieved using some hardware and software. A first demo of the software has been developed on 22 December 2019 and is available at <https://github.com/mth128/AudioEye>.

The idea is to convert images and video into sound. This is not an entirely new concept, as it has already been launched using the vOICe “Seeing with sound” app, proving this idea has potential. However, the philosophy and approach of Audio Eye is slightly different. The goal of the vOICe is to generate 2D images from left to right, with a sound sample of 1 second for each image. The idea of Audio Eye is to make it possible to track light sources and/or shapes in real time speed. If it works, the blind person may be able to catch a ball.

The way this is achieved, is by using a spiral shape (named the “Eyeweb”), in which the center of the spiral has the highest resolution of the image. The center of the spiral is converted to the highest frequency audio. The further away from the center, the lower the audio frequency. Every 360 revolution in the spiral the frequency is divided by a half. Hence the sound tone is fixed for each direction. The tone on the right is set to a C of a musical instrument. Every 30 degrees clockwise, the tone is one higher. Hence having all 12 tones in a single 360 revolution.

The idea of using this spiral, is in order to mimic the function of the eye. The center of the eyes retina has the highest pixel density, while the outer edges of the retina are mostly to provide context. If anything prominent happens in these edges, the person would move their eyes toward that prominent event. The same would be required for this spiral shaped EyeWeb: in order to get more detail, the camera must be aimed at the point of interest.

## Software

The Audio Eye software is available at <https://github.com/mth128/AudioEye>. It is available under the GNU 3 license. As for December 2019 the software is a C# demo module, which asks for an image. Currently it is not yet possible to load video’s or connect a camera. The software currently consists of:

* a toolstrip (currently only containing a load button for opening an image),
* a set of variables and settings,
* a picturebox, displaying the loaded image, together with an eyeweb (see next paragraph), located at the cursor position,
* three smaller images, showing the resulting grayscale images captured by the eyeweb, in the order: left ear, mono sound, right ear,
* sound that corresponds to the captured eyeweb, as explained in this document.

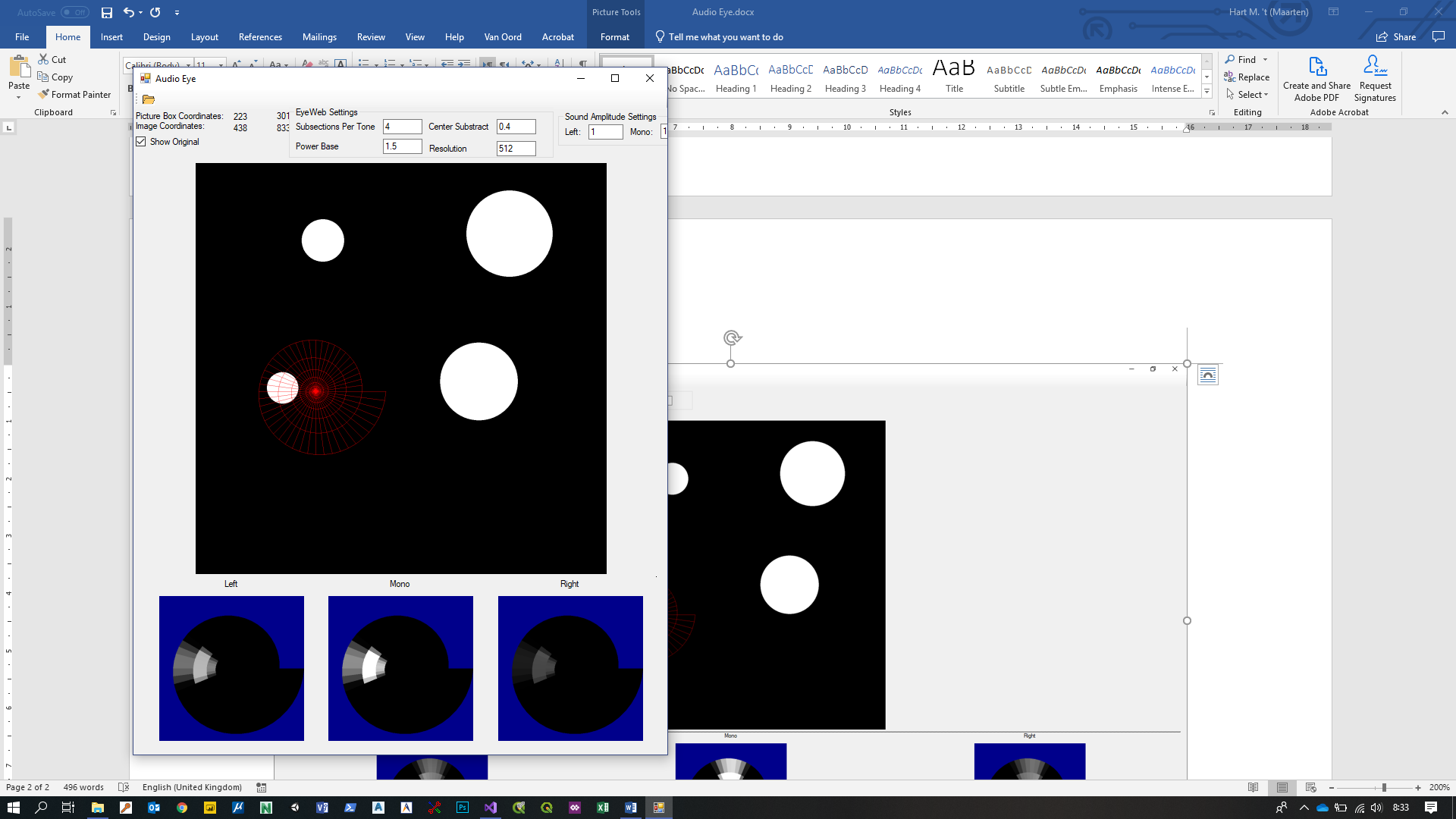
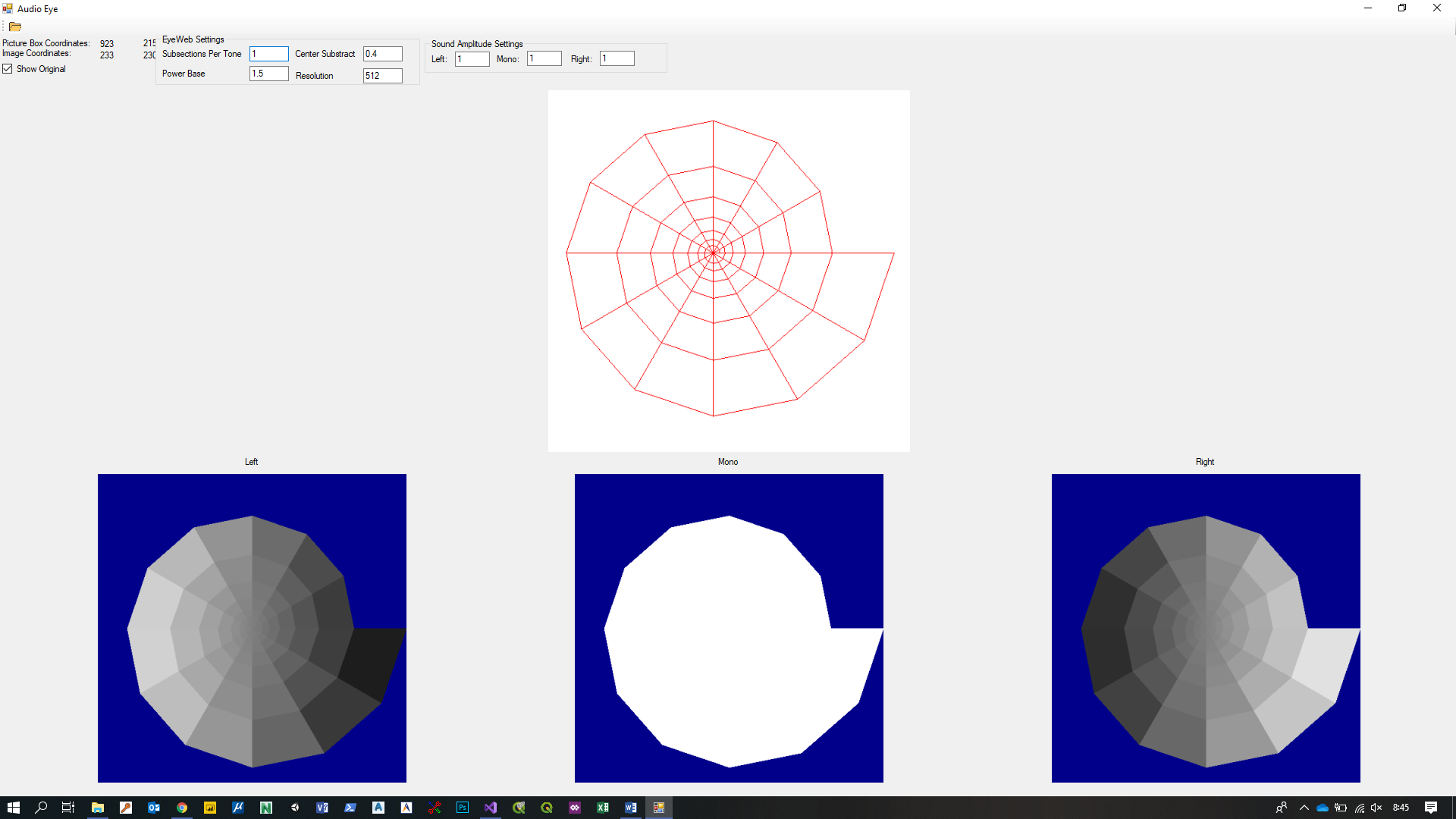


Image 1: a screenshot of the AudioEye software.

### EyeWeb

The core of the software is the “EyeWeb”. The EyeWeb is a 2D spiral shape, meshed into quads. The EyeWeb is positioned over an image, and for each quad a grayscale intensity value is extracted from the image. Each quad has its own sound frequency assigned to it. The sound frequency is basically a musical note (A to G#), which defines it’s direction. The octave defines the amount of inward revolutions.



C (523.2  
 Hz)

C (261.6 Hz)

C (130.8Hz)

B (123.5Hz)

A# (116.5Hz)

A (110.0Hz)

G# (103.8Hz)

G (98.0Hz)

F# (92.5Hz)

F (87.3Hz)

E (82.4Hz)

D# (77.8Hz)

D (73.4Hz)

C# (69.3Hz)

C (65.4Hz)

Image 2: The frequencies for each full-tone quad.

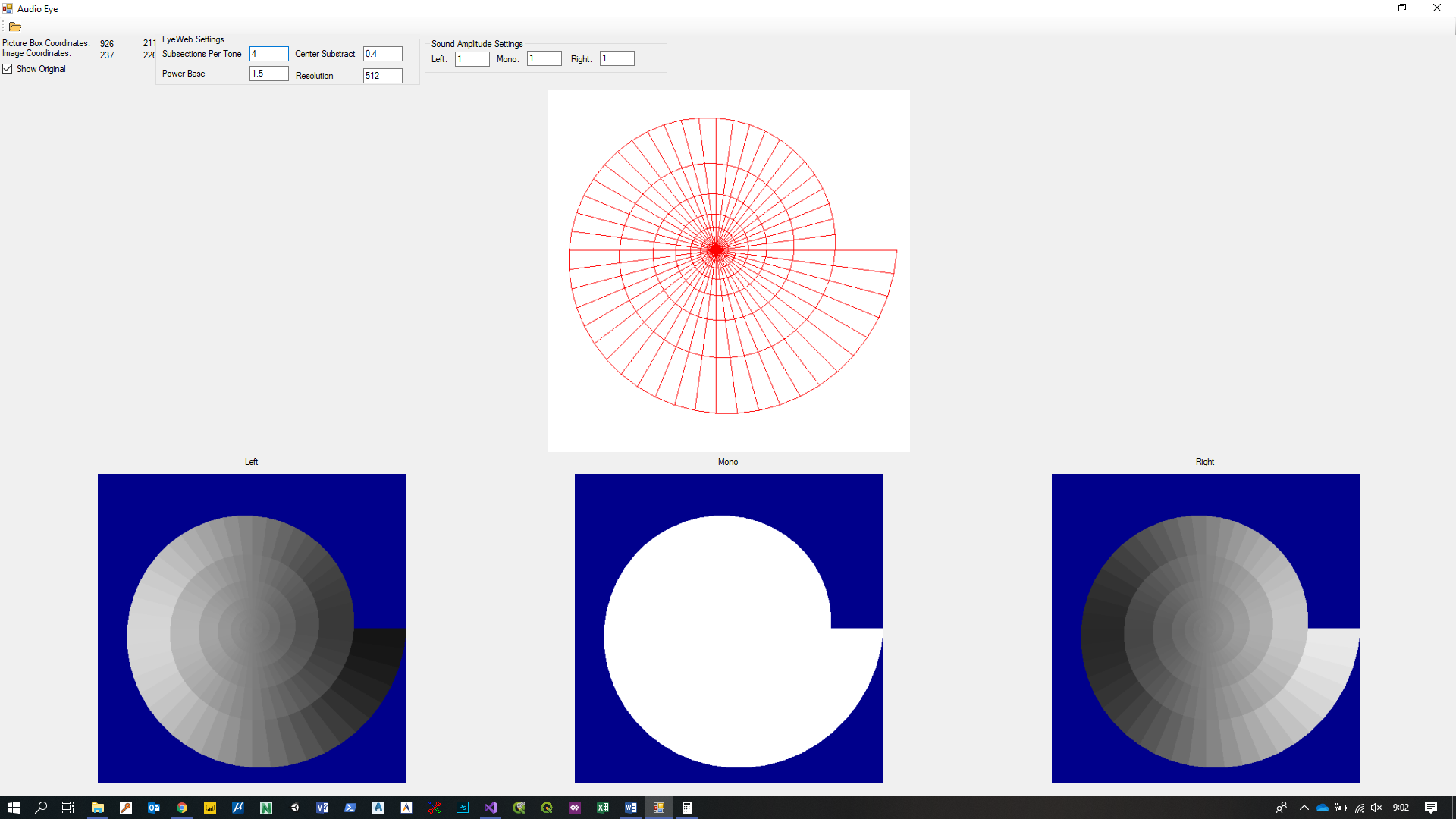


Image 3: A subdivision of the EyeWeb into 4 frequencies per tone.

The resolution can be further enhanced by subdividing the quads into smaller sections. Theoretically the web can be subdivided into an extremely high density grid, however this will be impractical for both performance and quality reasons. Performance because it will take more computer power to calculate the corresponding sound, but more important is the quality. The amount of small frequency bandwidths will eventually smear out into a vague sound, because of resonances occurring all over. Some small tests show 4 frequencies per musical note may be optimal for practical use, however further testing with other amounts is recommended.

If we consider note C of the lowest octave as tone 0, C# of the lowest octave as tone 2, and D of the lowest octave tone 3, we can calculate the frequency for each note. We set the frequency of C of the lowest octave to 65.4 Hz. The frequency of each tone can be calculated using the formula:

*Frequency = 65.4 \* 2tone/12*

This formula can be used for non-integer tones as well.

### Mono capturing

The default for the software is to capture the images into stereo sound. However, for the sake of completeness, this paragraph explains the method of mono capturing.

The source image is first converted into grayscale. This process is currently done using a very standard algorithm, however we are aware that improvements should be made in order to strongly enhance the contrast.

The EyeWeb is scaled to fit into a square image (defaulted to 512 x 512 pixels). The center of the EyeWeb spiral is exactly at the center of the square image. Then for each pixel is determined by which quad it is contained. Every pixel can be in at most 1 quad. So at this point there is no antialiasing, and it is expected that antialiasing will not increase sound quality.

This square image is overlaid over the source image at a given position. Currently the mouse cursor defines the position of the EyeWeb. Eventually this should be set to the position where the observer is looking at. Then the source image is cut to capture a corresponding square image.

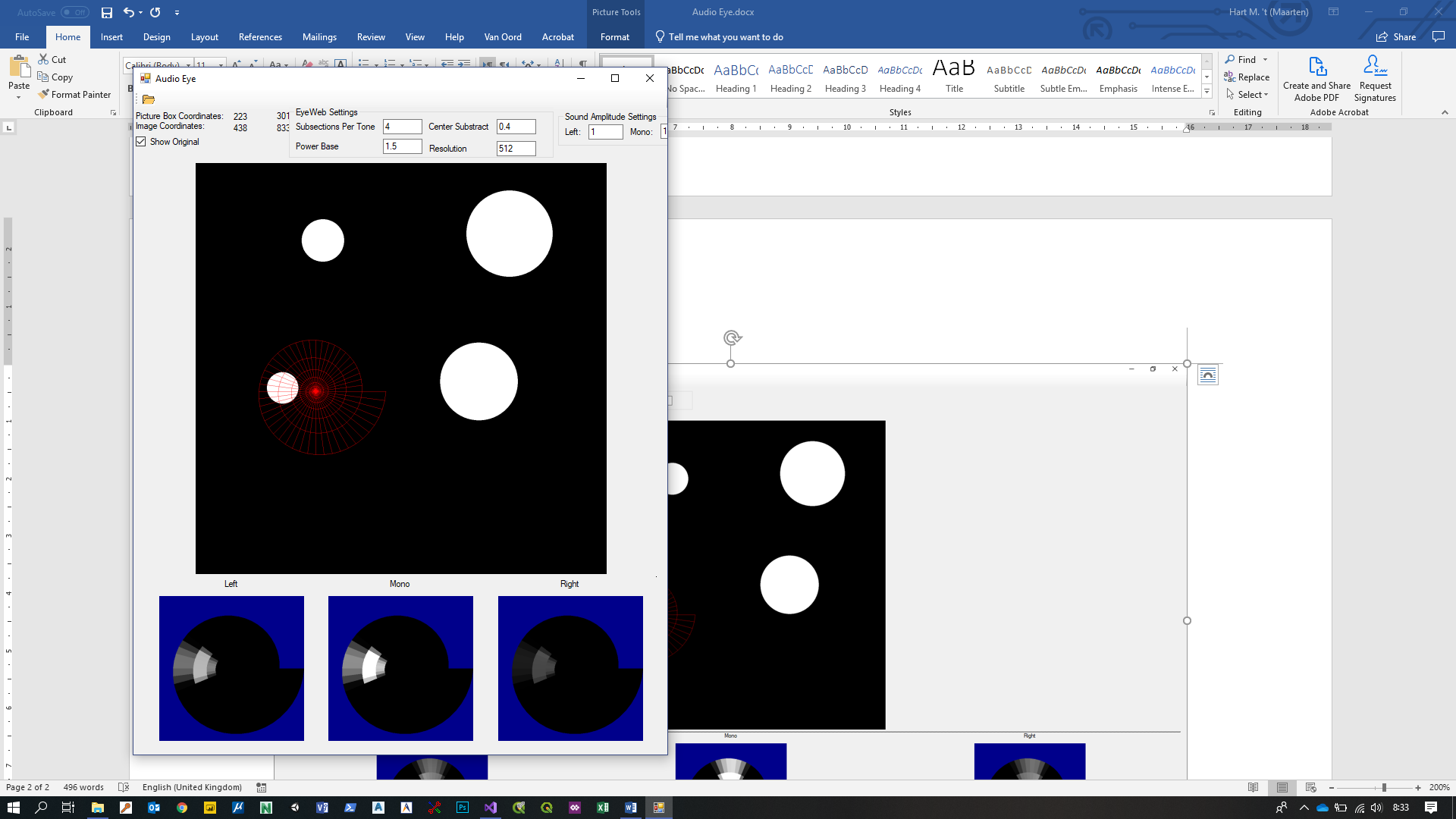
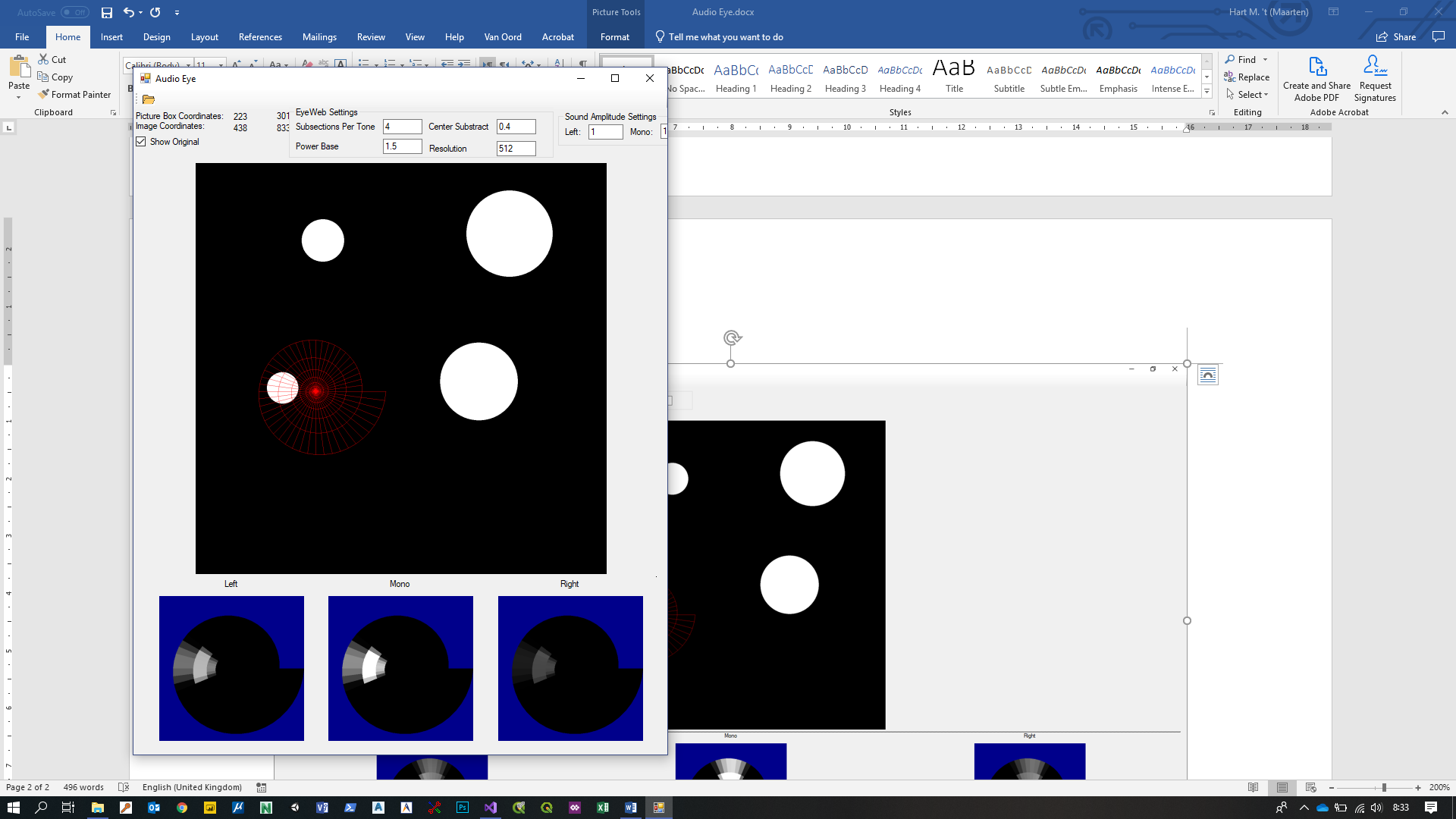
 

Image 4&5. Left: the EyeWeb spiral overlaid over an image. Right: the captured snapshot of the mono signal.

Finally, the square cut-out of the source image is used for defining the mono quad intensities. The intensity value of a quad is defined by the average intensity of all the pixels within that quad.

### Stereo capturing

Proper stereo can be achieved by using two cameras. However, some useful stereo effects can be achieved with a single camera as well. The mono intensity value for a quad can be calculated by calculating the average intensity for each pixel that is within the quad, as described earlier. A stereo effect can be achieved by weighing the horizontal position of the pixel into the calculation. The more a pixel is positioned to the left, the more it contributes to the intensity of the left image capture, and the lesser it contributes to the intensity of the right image capture. Of course, if a pixel is positioned more to the right, the opposite occurs. Applying this for each pixel rather than for each complete quad, contributes noticeably to the stereo effect: if an object is placed entirely within a quad, the mono signal does not show where the object is within that quad. The stereo effect however, shows a slight shift when the object is shifted within that quad.

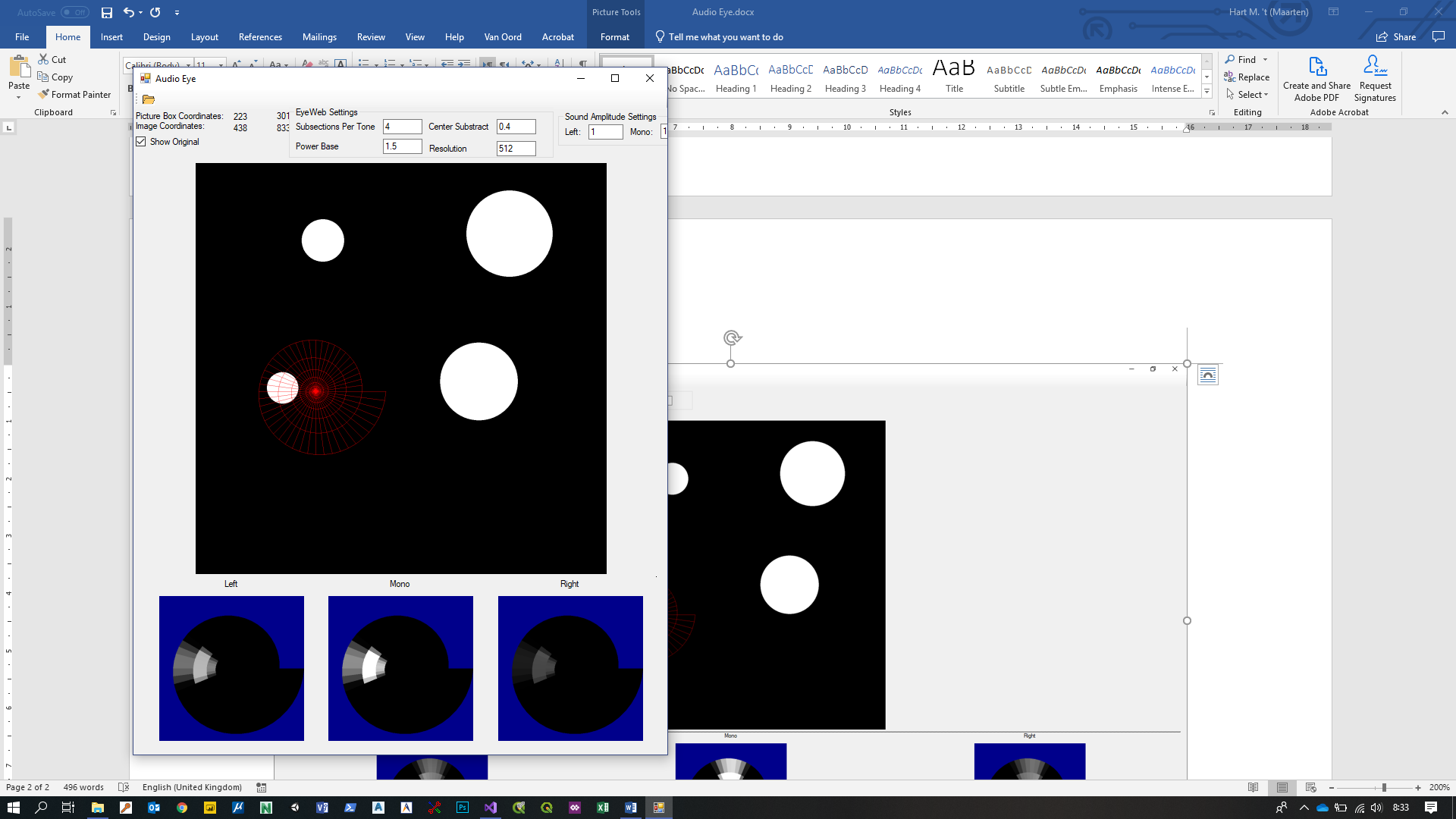


Image 6: the stereo effect of a single image capture.

### EyeWeb Variables

Besides the resolution and the subsection per tone (as described earlier), there are 2 other variables that can modify the shape of the EyeWeb:

* The Power Base
* The Center Subtract

The Power Base defines the size increase of the spiral arms. The “Power Base” is raised by the amount of 360-degree revolutions. Hence:

SpiralArmDistanceToCenter = PowerBaseRevolutions

The PowerBase should not be outside the range from 1.1 to 2. The most natural would be to use a PowerBase of 2. If we do that, the longest side of the quad matches the length of a single sound wave the quad uses. However, the center of the spiral becomes so small that some quads may not even contain pixels. For practical uses the PowerBase should be smaller. Currently the PowerBase is defaulted to 1.5, which is probably closer to optimal. The exact optimal value may be found with many experiments

The Center Subtract is an extra value to decrease the size of the center. This is particularly useful when the PowerBase has lower values. The Center Subtract should be defined by testing empirically. The default value is currently set to 0.4, which may be optimal with a Power Base of 1.5.

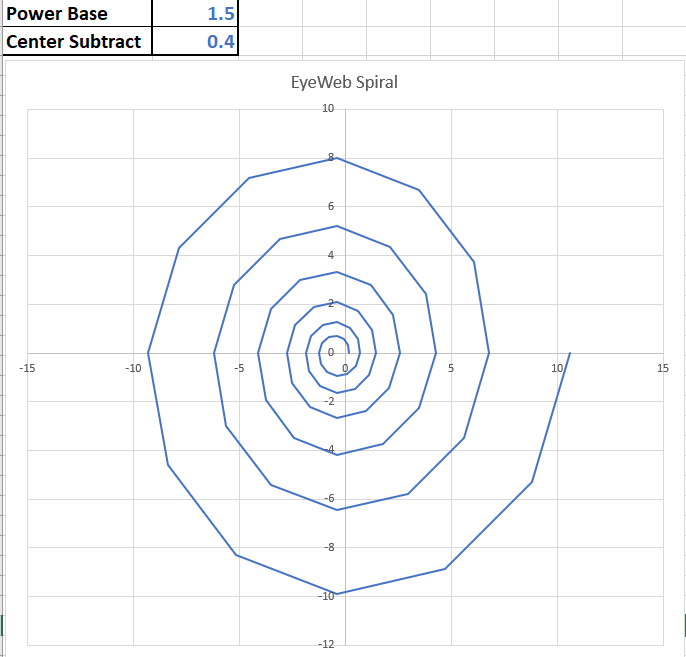
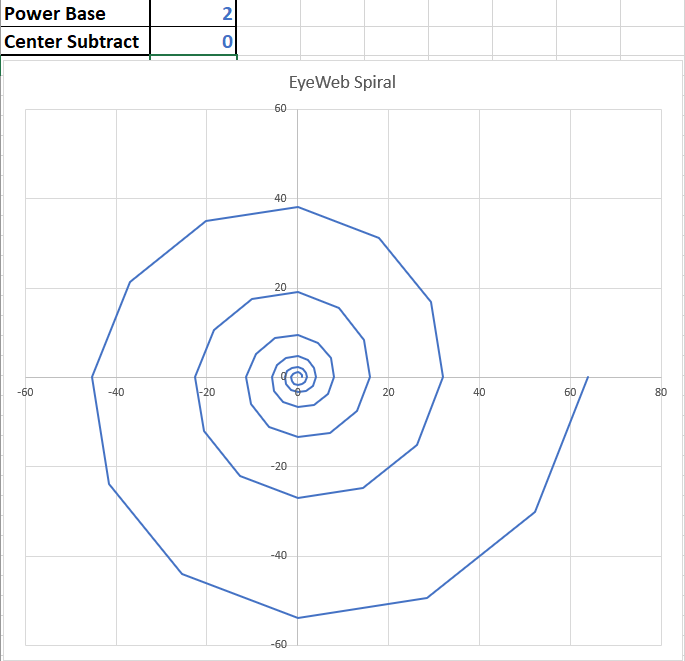
 

Image 7 & 8: the effect of changing the Power Base

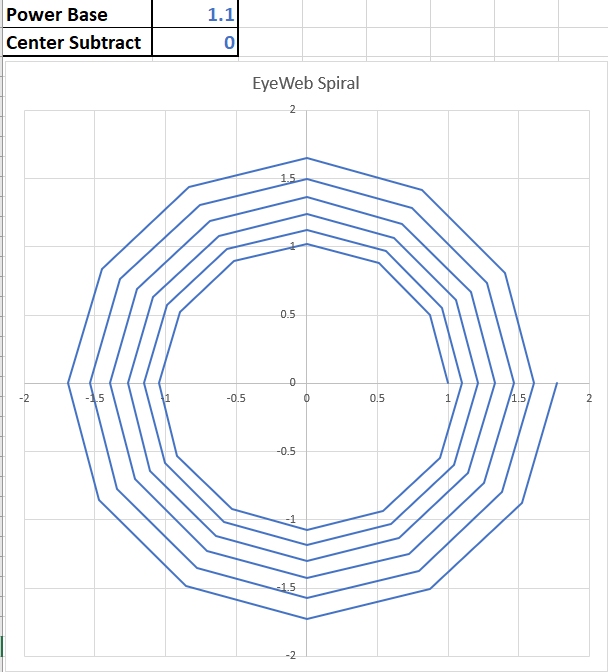
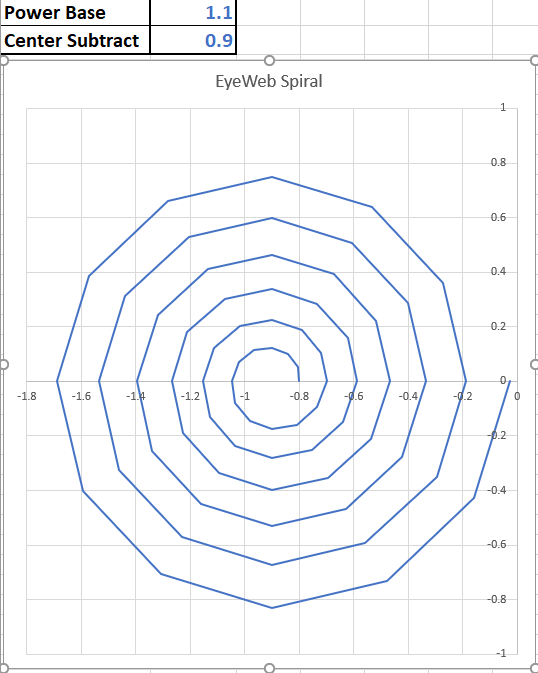
 

Image 9 & 10: the effect of changing the Center Subtract values (at a low Power Base)

The spiral shape is meant to mimic the properties of the eye: the center can see more detail than the edges.

### Playing with contrasts

In order to make the device useful, we should look for the most practical and comfortable image processing to reach the goal of properly hearing the image. The optimal settings for sound-image processing will probably differ a lot from the optimal settings for a visual image. The most obvious issue is to enhance the intensity of the image. However, other techniques could be used as well. In this paragraph we will discuss a couple of them.

The most important elements of an image for a blind person to know are the contours of objects. A very simple way to get the contours can be achieved using the following method:

* Use the original image as layer 1.
* Duplicate layer 1 into layer 2.
* Shift layer 2 two pixels to the right
* Duplicate layer 1 into layer 3.
* Shift layer 3 two pixels up.
* Calculate the (absolute of the) differences from layer 1 and 2 into layer 4.
* Calculate the (absolute of the) differences from layer 1 and 3 into layer 5.
* Add layer 4 and layer 5 into the final image.
* (Trim the 2 outer pixels of the image in order to remove the white border.)

Note that this method should (in most cases) be applied *before* turning the image into grayscale.



Image 11 & 12. Left: the original image. Right: the contours of the left image, by applying the described contouring method, using Photoshop.

Image 13 & 14. Left: the original image. Right: the contours of the left image, by applying the described contouring method, using Photoshop.

Since the advantage of this technique over that of the vOICe is the higher framerate, a more important goal is to detect moving objects, rather than defining the shape of objects. In order to find moving objects, we can use a similar technique as the contouring method described above:

* Use the previous video frame as layer 1.
* Use the current video frame as layer 2
* Calculate the difference between layer 1 and layer 2.

This method is a bit too simple, as this would detect too much as soon as the camera moves. So probably some filtering should be applied. This can be achieved by shifting layer 1 around in all directions, to find the “best fit” of layer 1 over layer 2. A potentially better way is to use a camera mounted gyroscope (which is available in many mobile phones), to find out how much layer 1 should be shifted to make it match layer 2.

## Hardware

This paragraph describes the optimal hardware. A simpler version with simpler hardware should be possible for testing purposes. The minimal hardware should be a camera and a speaker, which is usually available in a standard mobile phone. A simple increase in quality is to add headphones.

### Glasses mounted camera

In order to get real “vision” from AudioEye, is to use (sun)glasses with a camera attached to it. Probably a Bluetooth glasses camera + a mobile phone + headphones would make a minimal viable product, once the app is functional. There is no demand for high resolution cameras, as the lowest resolution camera’s available exceed the maximum achievable resolution by AudioEye. The old PAL resolution (640 x 480) is already sufficient. There is no problem using higher resolutions, except that it requires more computing power. Using this setup, the EyeWeb can be centered at the video being captured by the camera.



In order to get more 3D effect, it may be useful to have 2 cameras in the glasses: one for each eye.



If the blind person is capable of moving their eyes, it may be possible to add an eye tracking device to it. In that case the EyeWeb can be moved over the recorded video. If done properly, this may mimic true 3D vision to a blind person, albeit with a low resolution and no color.