Maarten ‘t Hart – 1 January 2020

# 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 software and hardware. A first prototype of the software has been developed on 22 December 2019 and the source code is available at <https://github.com/mth128/AudioEye>. A compiled version of this code is available at <https://sourceforge.net/projects/audioeye/files/AudioEye_Windows_01jan2020.zip/download>

The idea is to convert images and video into sound. This is not an entirely new concept, as there already exists an app the vOICe “Seeing with sound”. That proves this idea has potential. However, the philosophy and approach of Audio Eye is 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 moving light sources and/or shapes with a much higher frame rate, but with a lower resolution. Where the vOICe is image based, the goal of Audio Eye is to be video based. It is meant to be sufficient for a blind person to catch a ball.

The way this is achieved, is by using a spiral shape named the “Eyeweb”. The center of the spiral has the highest resolution of the image as were the outer parts of the spiral have a lower resolution. The center of the spiral is converted into the highest frequency audio. The further away from the center, the lower the audio frequency. Every 360 inward revolution in the spiral the frequency is doubled. A double frequency sound tone sounds like the same pitch, with a higher octave. 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 note higher. This results in that all 12 tones are represented in a single 360 revolution.

The idea of using this spiral, is 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 in order to see what it is. The same would be required for this spiral shaped EyeWeb: in order to get more detail, the camera must be aimed towards the point of interest. Where this point of interest is, can be heard by the main pitch of the sound.

## 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 for windows, 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 picture box, 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.

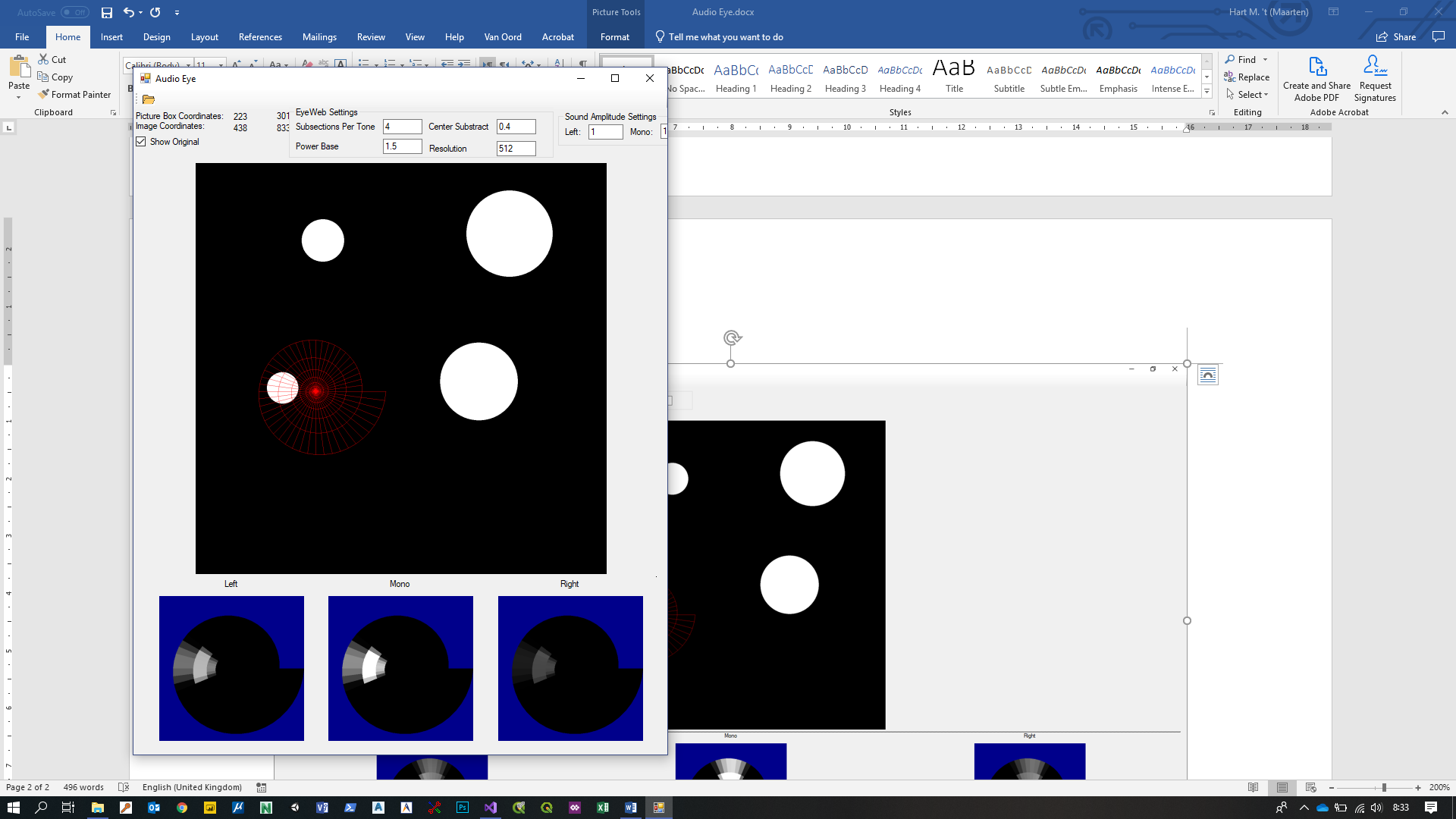
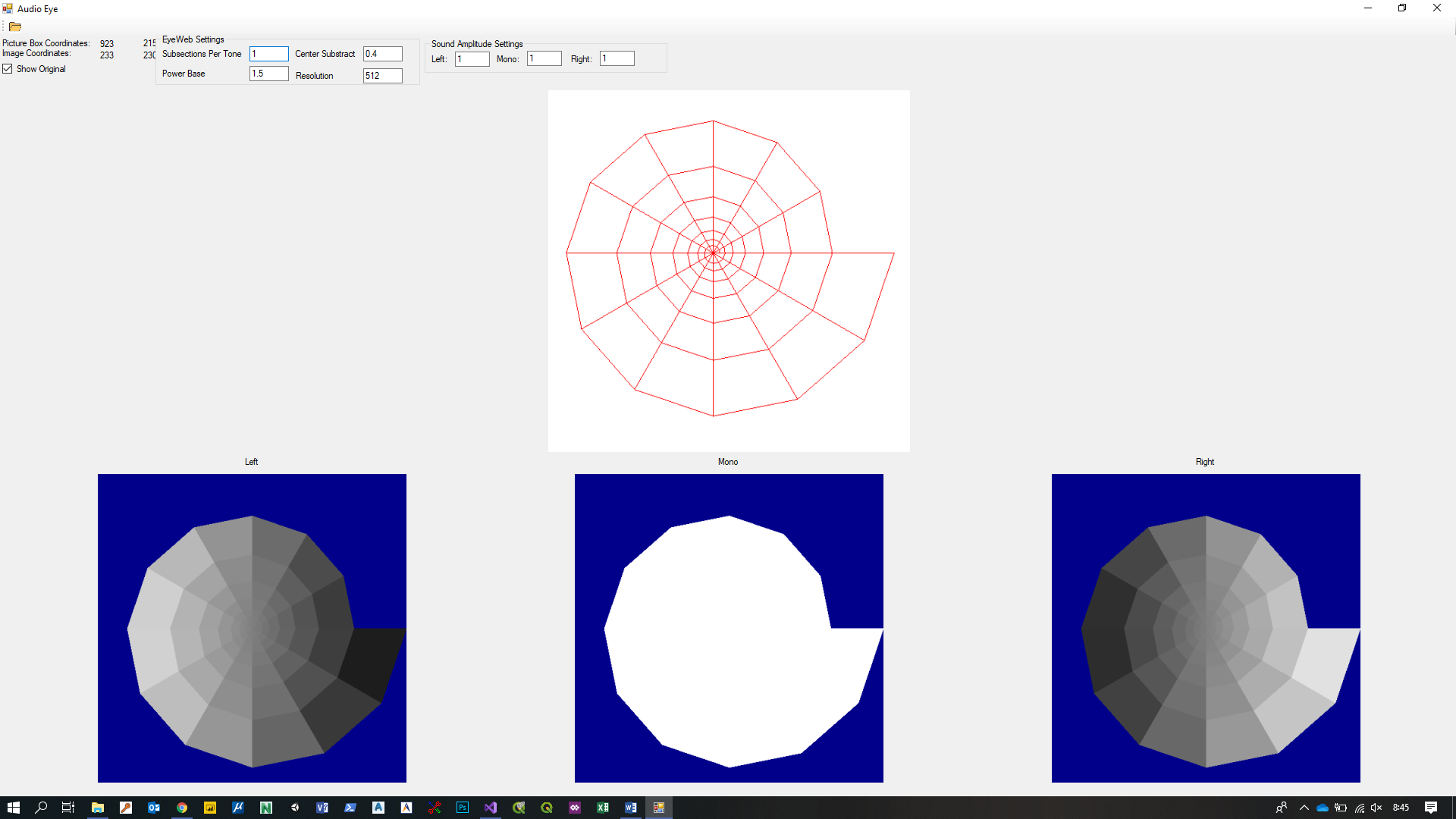


Image 1: a screenshot of the AudioEye software.

### 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.

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 the overkill in frequencies 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. It is recommended to use more than 1 frequency per note though, as 1 frequency per note makes the image sound like a badly played musical instrument, which is rather uncomfortable.

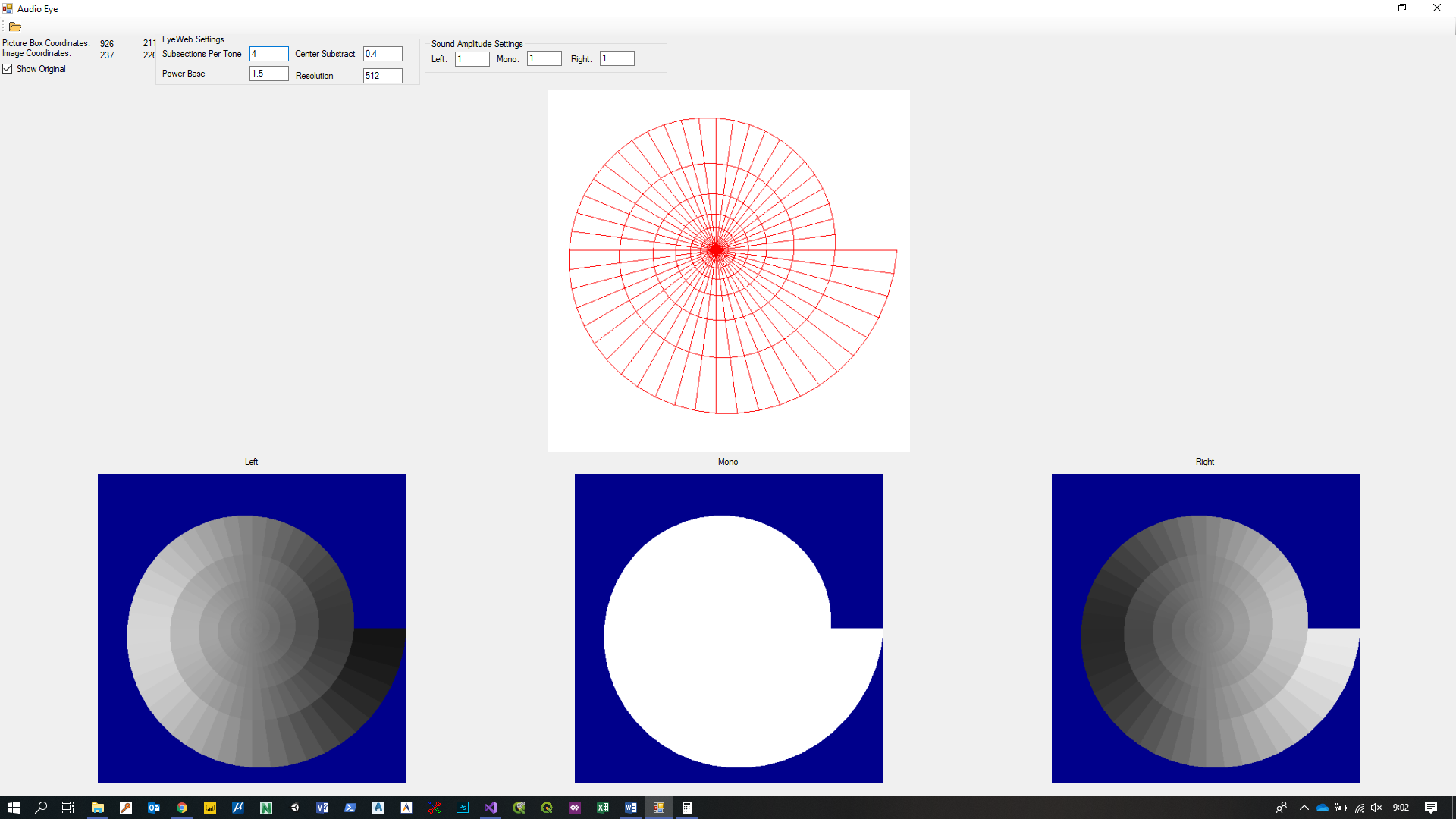


Image 3: A subdivision of the EyeWeb into 4 frequencies per note.

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, using the tone index as a variable. 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. Currently the lowest used frequency is set to 65.4 Hz and the highest frequency is set to 8041.5 Hz (8 full octaves). With a subdivision of 4 frequencies per note, this makes 12\*4\*8 = 384 sound pixels. The range could be further increased from 20Hz to 20000Hz, and potentially 8 frequencies per note. That would set the theoretical maximum to 952 pixels. However, getting into very high frequencies would make it impossible to hear the most detailed parts for some (mostly older) users. It would potentially also be uncomfortable for those who can hear high pitches. On the other hand, looking directly into a light source is also uncomfortable for sighted people.

Although these are not many pixels, it is expected that a trained user may be able to recognise a single letter in a very high contrast in a large font. This is expected, because some tests show that the snapshot images (later described in this document) do show distinguishable letters when the cursor is moved over them. If it can be seen in the snapshot, it can theoretically be heard 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 (or video frame) is first converted into grayscale. This process is currently done using a standard algorithm. We are aware that improvements to that algorithm should be made 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. Each pixel is then assigned to the quad that contains that pixel. Every pixel can be in at most 1 quad. This implies that there is currently no antialiasing. It is expected that antialiasing will not increase sound quality though.

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

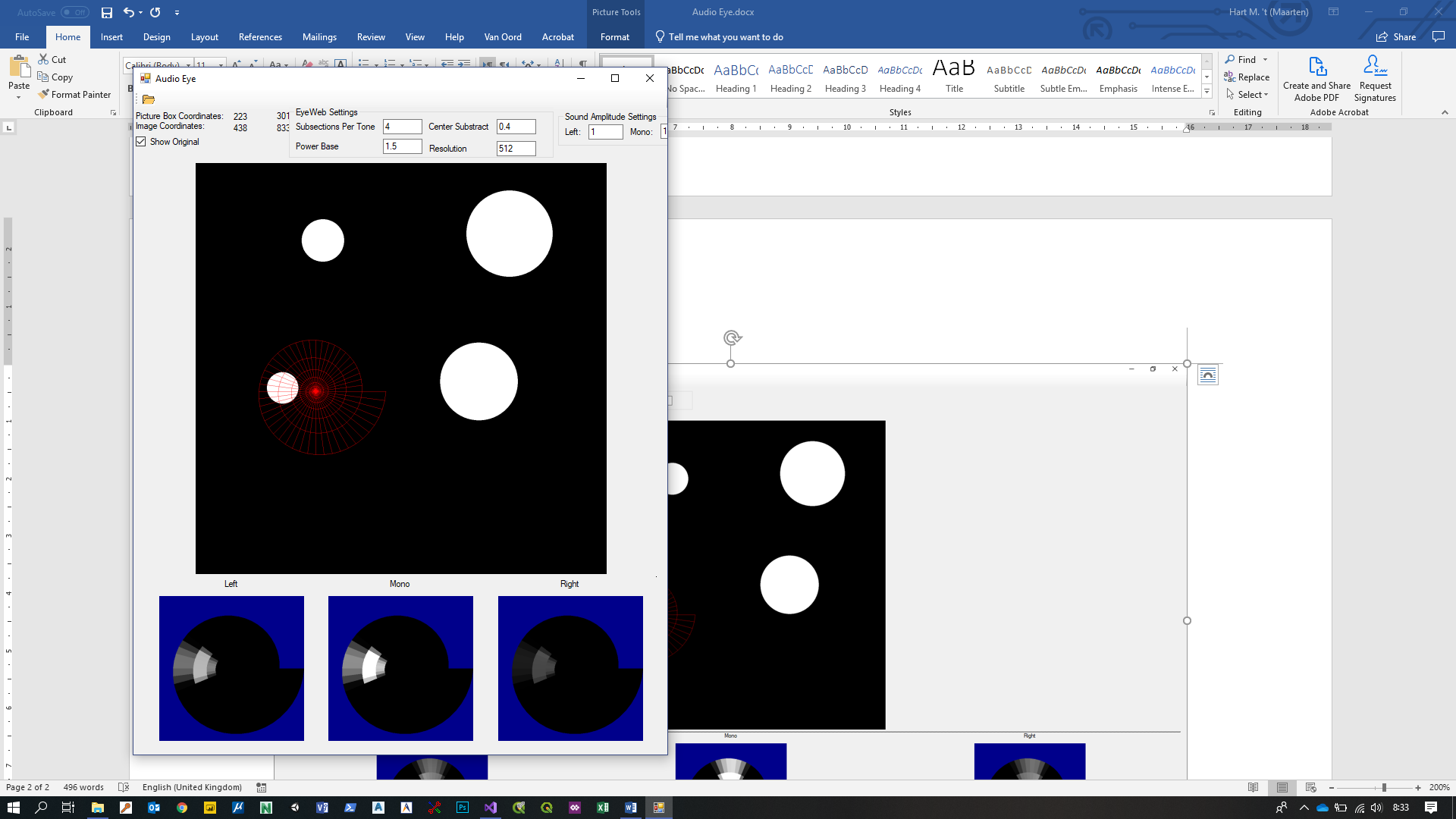
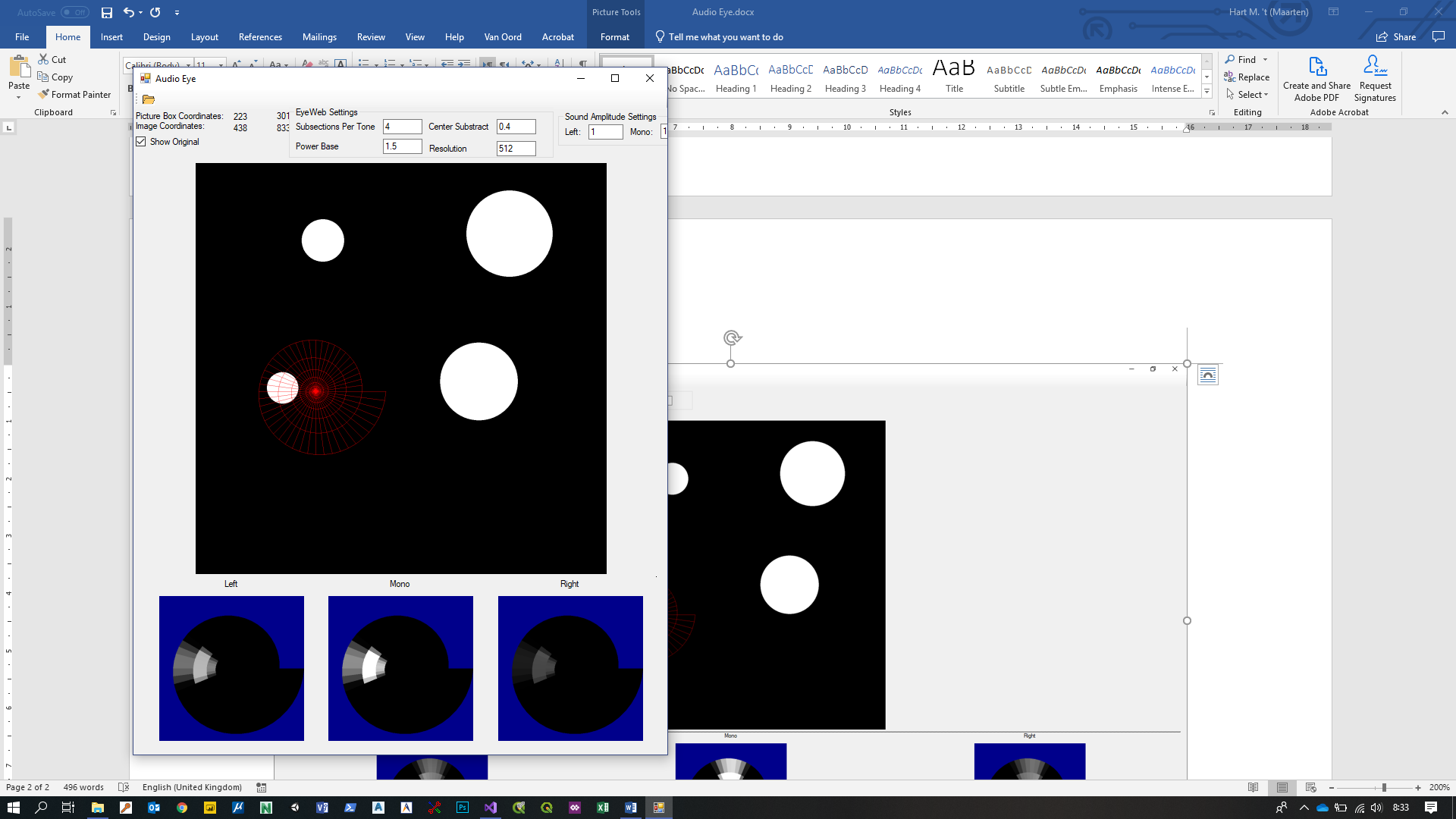
 

Image 4&5. Left: the EyeWeb spiral overlaid over an image. Right: the captured snapshot of the mono signal.

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: one for each eye. 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 in the mono paragraph). A stereo effect can be achieved by weighing the horizontal position of the pixel into this 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.

This is applied for each pixel, rather than for each complete quad. This contributes noticeably to the stereo effect: if an object is entirely contained by a single quad, the mono signal does not show where the object is within that quad; the mono signal does not change when the object moves within that quad. The stereo effect however shows a slight shift when the object is shifted within the quad.

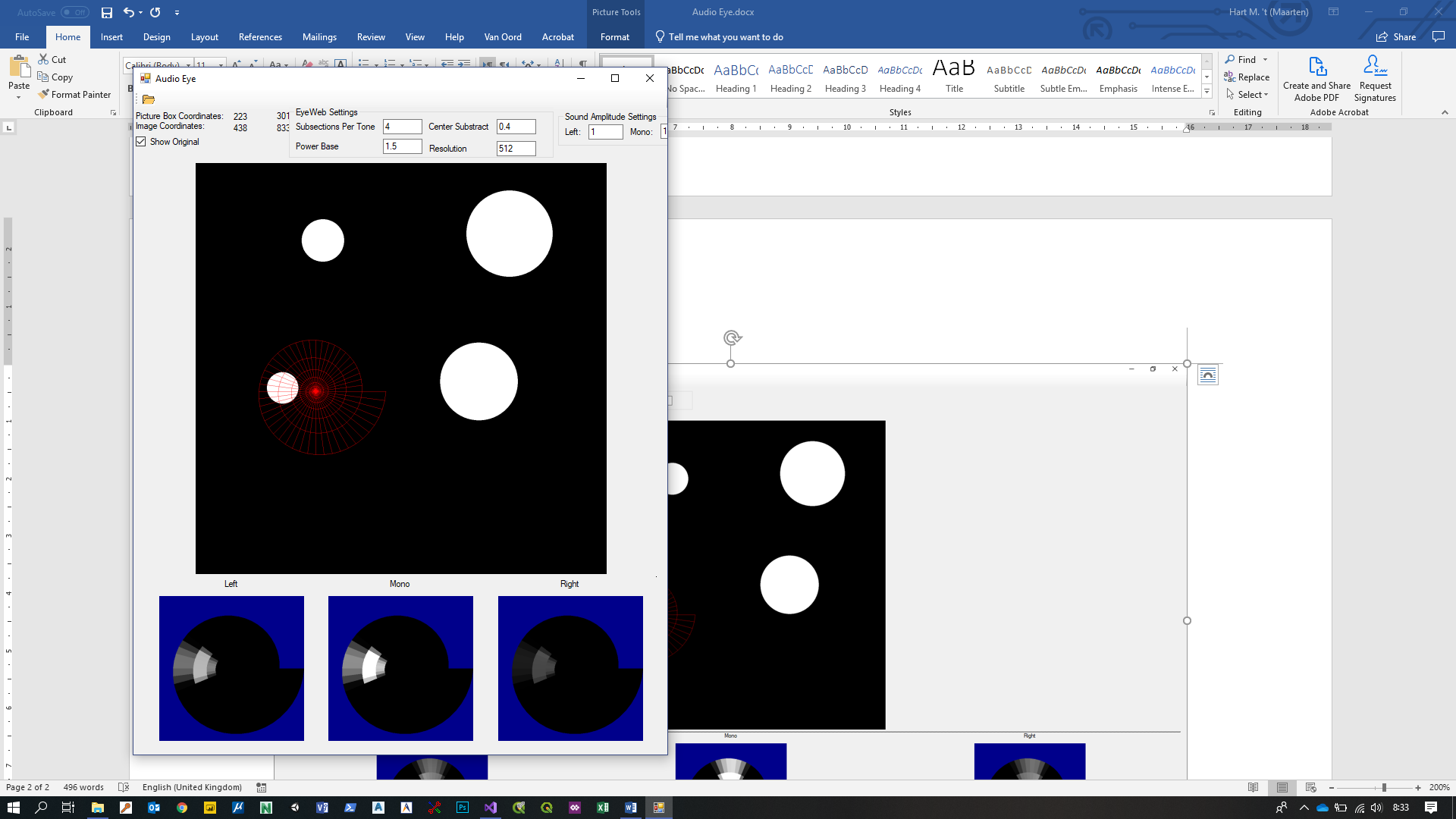


Image 6: the stereo effect of a single image capture.

### 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. In that case a double sound-wavelength corresponds to a double quad length. However, the center of the spiral becomes so small that some quads may not even contain pixels. For practical uses the PowerBase should therefore be smaller. Also, if we consider the area of the quad, rather than the length of a quad, the more natural value may be the square root of 2. Currently the PowerBase is defaulted to 1.5, which is probably close to optimal. The exact optimal value may be found experimentally. (It might even be desirable to keep this variable modifiable, to allow the user to zoom in into details. In that case a physical slider is more useful than the currently used input box.)

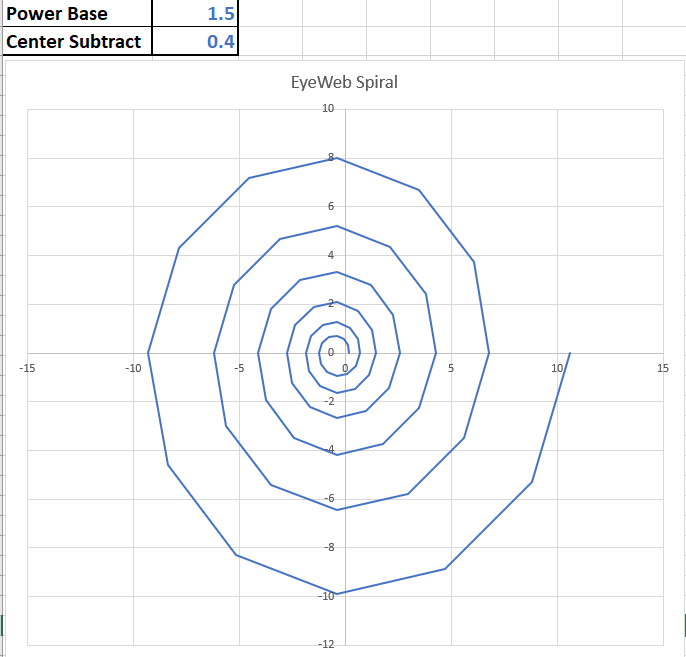
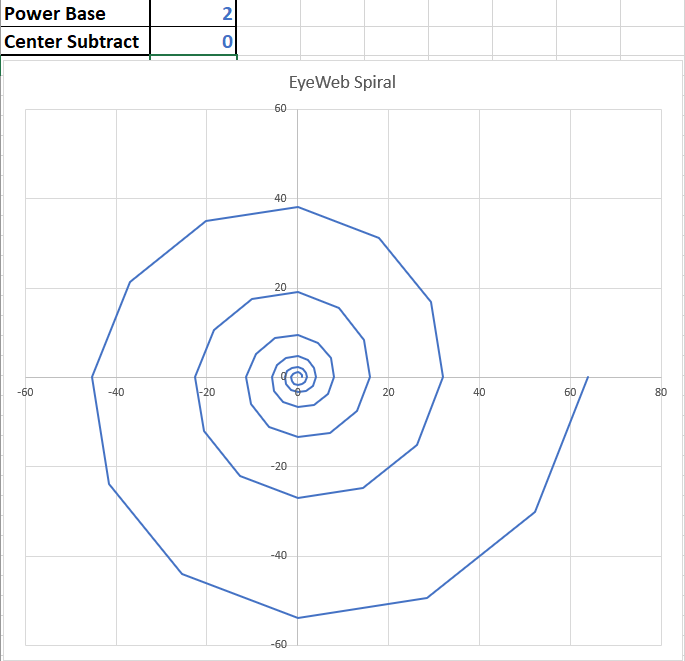
 

Image 7 & 8: the effect of changing the Power Base

The Center Subtract is an extra value to decrease the size of the center. This is particularly useful when the Power Base is set to a lower value. With a low Power Base value, the center of the spiral becomes very large relative to the spiral arm width. To reduce that, the center subtract must be increased. The optimal center subtract value varies per Power Base. The default value is currently set to 0.4, which is considered optimal with a Power Base of 1.5.

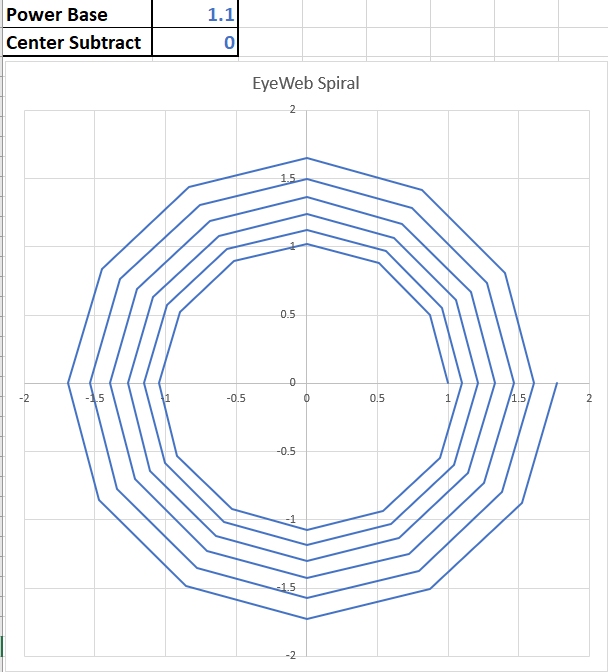
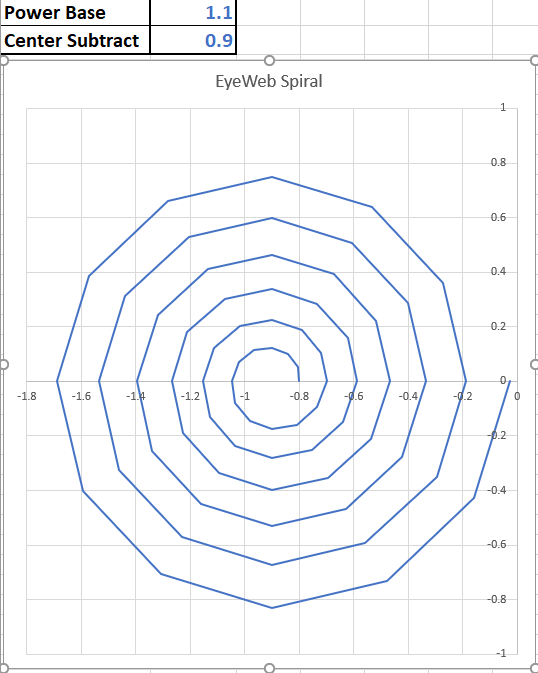
 

Image 9 & 10: the effect of changing the Center Subtract values (at a low Power Base)

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

### Playing with contrasts

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 contrast 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 within an image 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.
* Use the sum of layer 4 and layer 5 as the resulting image.
* (Trim the 2 outer pixels of the image to remove the white border if necessary.)

Note that this method should 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 goal for AudioEye is to see 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
* Use the difference between layer 1 and layer 2 as the final image. This shows the moved objects only.

Although this is the basic concept for detecting moving objects, this is too simple. The problem is that this would detect everything as moving object when the camera moves. Some correction must be applied for this. This can be done by using either software or a gyroscope to detect the camera movement.

## Hardware

This paragraph describes the optimal hardware. A simpler version with simpler hardware should be possible for testing purposes. The minimal hardware for testing should be a camera and a speaker, which comes standard within a mobile phone.

### Glasses mounted camera

To get real “vision” from AudioEye, one should use (sun)glasses with a camera attached to it. Glasses with a camera + a mobile phone + headphones would make a Minimal Viable Product (MVP). There is no demand for high resolution cameras, as the lowest resolution cameras available exceed the maximum achievable resolution by AudioEye. 640 x 480 pixels is enough. There is no problem using higher resolutions, although it requires more computer power. Using this setup, the EyeWeb can be fixed at the center of the video being captured by the camera. There is no need for moving the EyeWeb over the image. Sunglasses with cameras are not a new concept and are available on various web shops. Hence the hardware for the MVP is available to the general public.



To get more 3D effect, it may be useful to have 2 cameras in the glasses: one for each eye. Below is an existing product as illustration, however the cameras should be positioned in the center of the viewers eyes.



If the blind person can move 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. For this the hardware is not standard and should hence be developed. If done properly, this may mimic true 3D vision to a blind person, albeit with a low resolution and no color.