# Visual Effects

A crucial goal of this project is the implementation of a realistic simulation of what it means to be drunk or under the influence of other drugs using the technological toolset available at the time. Due to the focus on the experience and cooperation, we relied on the information provided by our industrial partner and did not go into scientific detail.

Many effects of alcohol and other substances such as balance issues, delayed reaction times and motor functions of course are important too, but these are not the focus of this project. Some of them may also be experienced when simply wearing a VR headset. This heavily depends on the individual as well as the duration during which it is worn though and is by no means calculatable.

## Information about Alcohol Effects on Vision Capabilities

Our industrial Partner ASN has provided us with information on the most essential visual effects of alcohol as well as other drug related substances. This information served this project as a basic element as we tried to implement as many of those effects as possible while also trying to keep the simulation as realistic as possible. The effects listed below are ordered based on the priorities we have assigned them in cooperation with ASN.

### Blurred View

**Similar to the increase in reaction times, it takes longer to capture and focus the surroundings while intoxicated. This can result in a blurry vision especially when the eye is not trying to actively focus on a particular object.**

### Inconstancy

All of the impairments mentioned in this chapter are not behaving in a predictable manner or always at the same ‘values’, but may change over time. It also is a possibility, that everything seems fine and clear as long as one remains sitting still. As soon as movement gets involved and the number of things to process drastically increases though, the substance related influences may start to manifest.

### Tunnel View

The human field of vision encloses usually 180 degrees. Sadly, you can only see sharp in a much smaller field of view (around 50 degrees) than that. Regardless of that, the peripherical vision is very important because it helps you react to warning colors or unexpected movements. Under alcohol influence these 180 degrees reduce steadily, being of course especially dangerous in overtake actions, spotting pedestrians, etc.

### Red Light Weakness

The ability to differentiate between different shades of red is very limited while drunk. This is especially dangerous regarding traffic signals and rear/break lights.

### Duplications

The human eyes both take individual images, which are then used to calculate the actual 3D image received. Under the influences of substances, this calculation can be off and result in certain items appearing to exist multiple times right next to each other.

### Flash/Night Blindness

Human pupils change their size based on the lighting circumstances. The time needed by the pupils adapt to changes thereof is typically longer under alcoholic influence and can lead to a sort of blindness for up to multiple seconds. This is period in time that easily can lead to a crash while driving.

## Image effects in Virtual Buzz

This chapter contains an explanation of the actual effects used within the app. Although the chapter naming is adjusted to match the individual components in our Unity project, the order in which they are explained matches the one used in the previous chapter (see 7.1).

### Blur

The blurriness of the overall image provided by the camera in the app was mainly achieved through the usage of two effects:

**Blur (optimized)** [5][6]**:**

This component implements a form of the “Gaussian smoothing” or “Gaussian blur” effect. This effect is rather common in image processing and primarily used to reduce the detail or noise of an image. This is achieved by selecting small groups of pixels in an image and then filtering out or smoothing the individual pixels with high contrast to the identified center to make the group as a whole appear more alike or unified. As mentioned, this can help with noise reduction so if e.g. an image had a few pixel errors (completely white) distributed over the image and this process was applied, the color of those pixels (but also others) would be changed to something more similar to their surroundings.

The overall reduction of detail within an image while still appearing somewhat close to the original is what we use this component for in our app. the option to change the following values:

* Downsample

This could be used to heavily reduce the image quality before applying the actual effect with the goal to be more efficient while losing more detail. We decided not to use this feature, since the image captured by the camera should still closely resemble the real world as well as possible and is not of exceptional quality to begin with.

* Blur Area

This parameter refers to the size of the pixel groups used in the smoothing process This value was the focus in our app, since it results in the most simple and efficient way to produce the desired blur effect. The user has the option to change the used value himself within the configuration.

* Blur Iterations

The amount of iterations defines the number of passes through the actual blur process. In an effort to optimize the performance and since the adjustment of the blur area already resulted in a sufficient result, we left this value at one iteration.

**Camera Motion Blur** [7][8]**:**

As the name indicates, this component is applying a form blurriness based on the direction as well as the speed at which the camera is moving around. It is achieved by calculating a trail for individual pixels from the previously displayed images visualized in a directional blur. The goal is to provide a smoother effect for swift movements instead of the single frames well suited for games with a first-person perspective.

In our app, this effect merely acts as a supplement to the default blur effect as well as the possible side effects from using a smartphone camera. We stuck with the default values provided for all aspects except for the velocity scale. That parameter influences the velocity that has to be reached before triggering the motion blur effect and to which extent it is applied. So, since we are in an AR environment and controlling the camera with our head, we cannot expect movements comparable to the ones with a mouse controlled and increased that value.

### Randomization [9] [10]

This component was entirely built to provide the experience described in the inconstancy of individual effects while under the influence of substances. It currently applies to the blur and tiltshift components, since those are where the action happens primarily.

The algorithm used for this is called random walk and partially based on the Brownian motion for particles. It essentially performs changes in small steps based on a function rather than assigning a completely new random value to the component. In our case, the probability for each step to remain on the same value is defined as 30%. The remaining probability of 70% is then distributed among the chances of increasing and decreasing while taking into account how close to a boundary the value is. So, directly after initializing the random walk, the distribution is 50/50 leaving the chance to increase the value at the same chance of decreasing (35%). As the current value gets close to one of the boundaries of the possible range, the chance to get even closer to the boundary gets smaller and reaches 0% if the value is equal to it. With the current configuration, the changes potentially happening on each step are barely noticeable, but there is a very noticeable difference between the lower and upper end of the possible range.

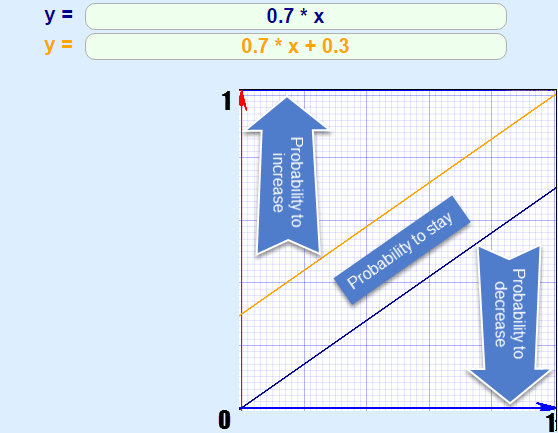


Abbildung 3: Graphical Visualization of the used Random Walk

### Tilt Shift

By wearing a headset such as Google Cardboard and solely looking through the camera of your smartphone already acts as a form of tunnel view compared to our usual vision. This component is used to create a form of an additional tunnel vision by blurring out the parts of the image based on their distance to the center. This is meant to closely resemble that limited ability to focus while under the influence of substances.

The tilt shift component in Unity comes with the following options for configuration:

* Mode

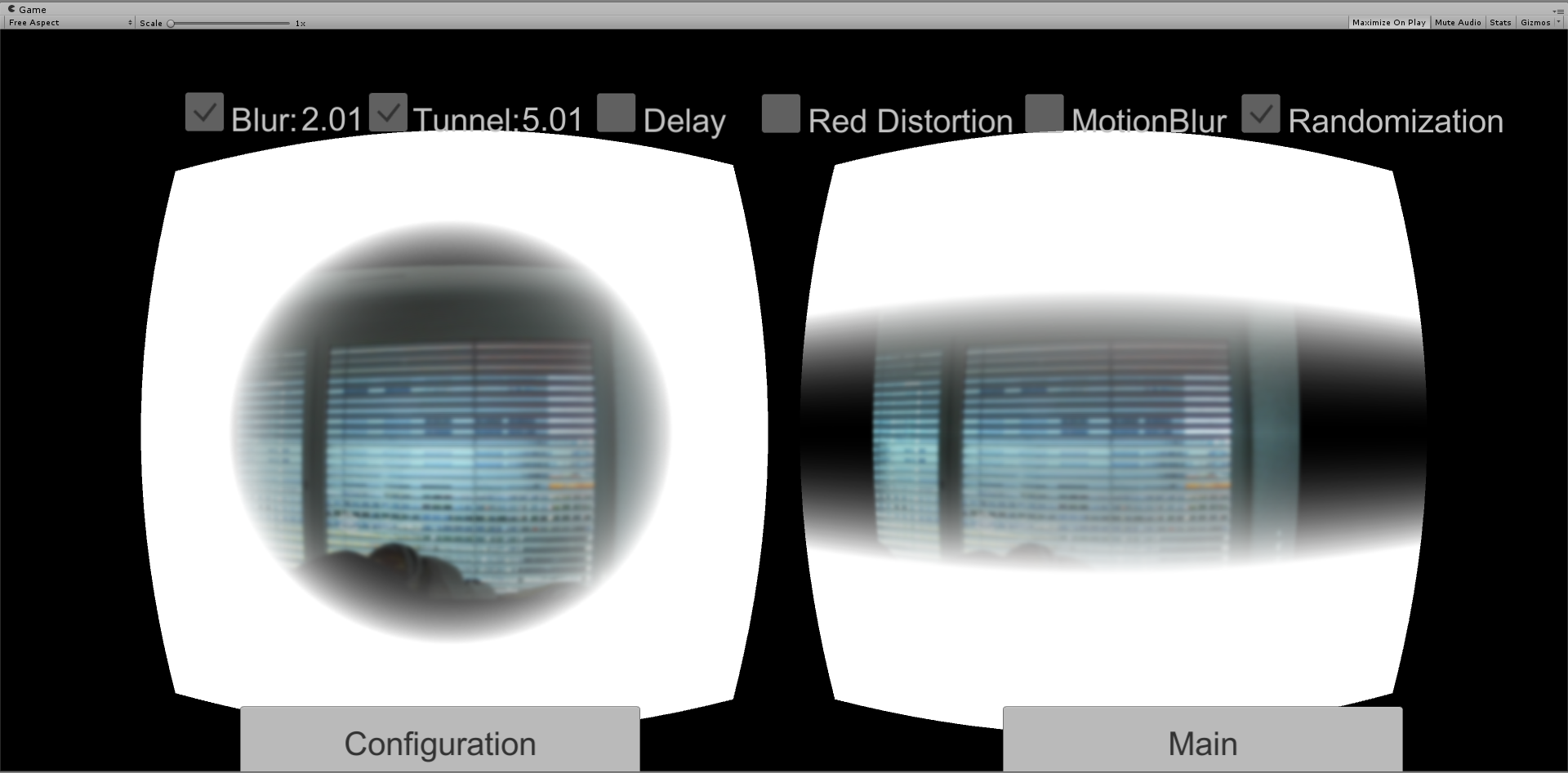
This setting defines whether the distance to the center of an image used for blurriness should be measured vertically or radially. So, the concrete options available are called “vertical defocus” and “radial defocus” and their calculated blur areas can be seen in the following picture:

Abbildung 4: TiltShift Modes preview

We considered both of these modes viable, but ended up using the vertical defocus, since the effect is only supplementing the already limited field of view within the AR app.

* Quality

This is referring to the number of samples taken to calculate the individual blur effects. Because this would affect performance when using high instead of normal quality and we use the effect in combination with a regular blur anyways, we went with normal.

The third option is simply made for preview and renders only the blurred areas as displayed in the image above.

* Max Blur Size

The maximum blur distance is defined by this setting and we simply used the maximum of the available values in our configuration.

* Blur Area

This refers to both size and strength of the blur applied in the designated areas. Aside from the regular blur effect, this was the second point of focus in our app and this value is therefore exposed to the user in the configuration of the app.

### Color Correction

This Image effect tries to simulate the weakness of differentiating between the different shades of red. In order to obtain this visual effect, we use the Color Correction Curves[[1]](#footnote-1) feature from the Unity Image effects.

Color Correction Curves make color adjustments using curves for each color channel (RBG). Depth based adjustments allow you to vary the color adjustment according to a pixel’s distance from the camera.

Curves work on each of the red, green and blue color channels separately and are based around the idea of mapping each input brightness level (i.e., the original brightness value of a pixel) to an output level of your choice.

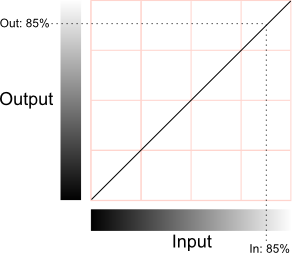


Abbildung 5: Mapping Options of Brightness for each Channel [11]

The horizontal axis represents the input level and the vertical represents the output level. Any point on the line specifies the output value that a given input is mapped to during processing.

In our case, we choose to edit the red channel curve and configured its output in a way that the visibility of the red color is reduced and looks more like black. As you can see in the image below, it is difficult to recognize whether a traffic light is actually illuminated (the light in the image is a bike brake light).

The collateral effect of this curve is the reinforcement of the green color, but ASN told us that this “green effect” complements well the red light weakness because this weakness is stronger during the night. And the green effect slightly darkens the whole vision, so this small side effect is considered acceptable.

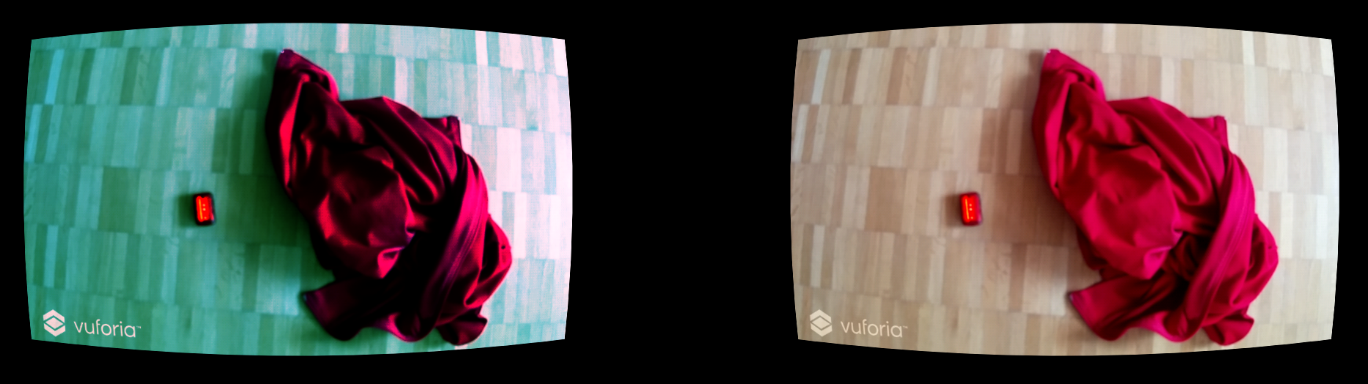


Abbildung 6: Preview of the used Red Color Distortion

### Delay

Depending on the smartphone used, merely using the running camera to look around already comes with a noticeable or even significant delay. This component was created by us to provide an additional delay and to partially acts as a proof of concept for our understanding of image effects in Unity.

This component is called after all the rendering and other image effects have completed. It contains a collection of frames, to which the current image is always added when it is called and enabled. Once that collection reaches a specified limit (in our case 15), the first texture of the collection is displayed on the running camera instead of the current texture. The specified limit was chosen by testing the feature on the smartphone used for this project (Nexus 5X) and creating a delay slightly bigger than a second.

1. https://docs.unity3d.com/Manual/script-ColorCorrectionCurves.html [↑](#footnote-ref-1)