DESIGNING A USER-FRIENDLY DESKTOP APPLICATION FOR CONDUCTING PSYCHOVISUAL EXPERIMENTS

GERBEN NOORDELOOS

Primary supervisor: Dr. Cara Tursun
Secondary supervisor: Prof. Jiří Kosinka

ABSTRACT

Psychovisual research aims to understand how our brains process sensory data and has led to many improvements in technical fields such as visual computing and signal processing. The existing programs that most psychovisual researchers use to set up their experiments are all difficult to learn and extend due to various factors. This thesis explains how we designed a user-friendly application that allows researchers to conduct psychovisual experiments without doing any programming themselves and how we ensured a modular approach towards the available stimuli and experiment methods. If psychovisual researchers started using this toolbox, we expect it would improve the efficiency at which they can carry out their research and make the field of visual psychophysics more accessible to researchers who are not familiar with software development.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

VR Virtual Reality
XR Extended Reality
GUI Graphical User Interface
URL Uniform Resource Locator
JSON JavaScript Object Notation
2AFC Two-Alternative Forced Choice
2AFCref Two-Alternative Forced Choice with reference

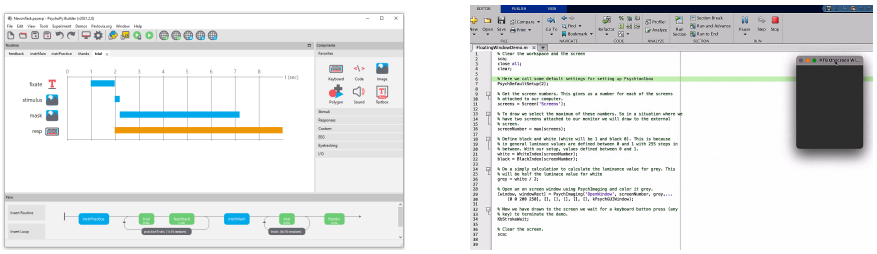
1 INTRODUCTION

Psychophysics is a methodology and research area that investigates the connection between the physical properties of stimuli and how we perceive them [1, 3]. Psychophysical research has led to improvements in many technology-related practices, the most noteworthy of which is visual computing. Modern examples include foveated rendering, which reduces the effort required to render areas in the peripheral vision for VR displays, and visual signal compression [7, 12]. Another notable beneficiary is in the medical field, viz., neurology [5]. Visual psychophysics focuses specifically on human vision, including both the eye and brain, and such research is nowadays often done by using computers to perform experiments to measure visual sensitivity. To carry out these experiments, researchers or students often use software (referred to as toolboxes) that can aid them with setting up experiments and storing and analysing their results. Two examples of popular toolboxes are shown in Figure 1.1.

Existing toolboxes are limited by several factors. Firstly, most of these toolboxes require their users to have programming experience in a specific language, as they need to write code to set up their experiments. Secondly, many toolboxes are very broad in their scope, for example focusing not just on visual psychophysics but also on psychoacoustics and other fields of perception. This broad scope leads to a higher amount of complexity, and a steeper learning curve than a toolbox focused solely on visual psychophysics could have. The increased complexity of such toolboxes also makes it harder to add additional features since the existing codebases are larger and harder to understand. Additionally, many toolboxes have been discontinued and are no longer maintained.

We address these issues by creating an open-source psychovisual toolbox that allows users to set up experiments without writing code. This toolbox will benefit researchers and students in the fields of visual psychophysics by allowing them to more easily and efficiently set up and carry out their experiments. Improvements in psychophysics made possible by our toolbox will also benefit others, such as people suffering from sensory disorders. We aim to make our toolbox as hardware-independent as possible to ensure the accurate reproducibility of experiments on different displays, including Virtual Reality (VR) displays.

This thesis outlines our approach to creating this toolbox and showcases its capabilities. In Chapter 2, we investigate the state of psychovisual research and the currently existing toolboxes. Chapter 3 provides



(a) PsychoPy (b) PsychToolbox

Figure 1.1: Two examples of existing toolboxes. PsychoPy's experimental "Builder" interface is shown in (a). (b) illustrates a sample script in PsychToolbox that is used to render an empty window that may be used for displaying visual stimuli. To accomplish only such a minor functionality, the user must write a significant amount of boilerplate code.

a general overview of our implementation, while Chapter 4 details the experiment methods and stimuli that we have implemented. In Chapter 5, we discuss the storage and analysis of results, as well as the results of an experiment we performed ourselves. We investigate how our toolbox can be adapted to work with VR displays in Chapter 6. Throughout this thesis, we specify two different types of people who may use the toolbox:

1. User: the user uses the toolbox to set up experiments and analyse their results.

2. Participant: the participant is someone participating in an experiment set up by a user.

2 RELATED WORK

Psychovisual experiments are performed by presenting stimuli on a computer display and using procedures such as the Two-Alternative Forced Choice (2AFC), two-interval forced-choice (2IFC), and yes-no tasks [4]. The 2AFC method presents the participant with two distinct stimuli: a base stimulus and a test patch where some parameter is changed, such as the contrast or sharpness of an image. The participant is then asked which of the two stimuli was the test patch. It is a forced-choice task, meaning participants must choose one of the stimuli, even if they see no difference. The 2IFC method is similar but presents the stimuli sequentially instead of simultaneously [2]. The yes-no task presents participants with a series of stimuli and asks them to determine whether or not a given phenomenon is present. Generally, the 2AFC and 2IFC methods are preferred over the yes-no task, as they reduce response biases [10]. The testing procedures can also be made adaptive to the observer's responses, resulting in less time spent on stimuli deemed too hard or too easy to see. Multiple toolboxes for designing and running psychovisual experiments already exist, which we discuss below.

2.1 PSYCHTOOLBOX-3

Psychtoolbox-3 is, with over 285000 downloads as of May 2018 [9], one of the most widely used software for psychovisual experiments. Psychtoolbox version 1 was first created exclusively for Macintosh devices in 1995 by David Brainard, who was, at the time, a professor at the University of California. The more recent Psychtoolbox-3 was released in 2007 and is still updated regularly, almost exclusively by Mario Kleiner. The current version is a MATLAB library containing functions for vision and neuroscience research, using OpenGL for rendering. It has VR support through its PsychOpenXR driver. MATLAB was chosen due to its ease of use for mathematical purposes, such as calculations and plotting graphs, and its many predefined high-level functions [9]. Other advantages of MATLAB include its platform independence, and its widespread use in the scientific domain, which has led to good community support. Nevertheless, the main disadvantage of Psychtoolbox-3 is that it requires users to program in MATLAB, which, while it provides users with a high degree of customisation, increases the time and effort needed to set up experiments (see Figure 1.1b) [6]. This requirement also makes it less accessible to people

unfamiliar with MATLAB, and people who do not own MATLAB, which is a commercial product.

2.2 PSYCHOPY

PsychoPy is a free, open-source software for behavioural science experiments. It is written in Python and makes use of OpenGL. PsychoPy was first created as a library in 2003 by Jon Peirce, a professor at The University of Nottingham. Peirce has been regularly providing updates to PsychoPy, including an editor and the Builder interface shown in Figure 1.1a. This Builder interface allows users to create graphical representations of experiments, for which it can generate and run scripts, meaning that users do not have to do any programming themselves [8]. PsychoPy has limited support for VR displays, as it is incompatible with the builder interface and only supports a handful of VR headsets. PsychoPy's main drawback is its complexity, which is caused by multiple factors. Firstly, PsychoPy has a long history of development that has resulted in a large codebase that is difficult for new users to make extensions to. Moreover, PsychoPy is intended to be used for all behavioural sciences, as opposed to just visual psychophysics, which has also increased the complexity of its codebase. This broad scope also makes it harder to use the builder, because it has made the GUI more complicated. Due to this complexity, new users must invest significant time and effort into learning to use PsychoPy.

2.3 OTHER TOOLBOXES

Whereas the previous two were focused mainly on performing experiments, other toolboxes exist that focus on the analysis of psychophysical data and analysis of psychometric functions, such as the Palamedes and psignifit toolboxes mentioned by Kingdom [3]. However, these toolboxes are suboptimal for similar reasons as PsychoPy and Psychtoolbox-3. For example, Palamedes and psignifit both require the use of MATLAB..

3 IMPLEMENTATION

We have created a toolbox that allows its users to perform psychovisual experiments without having to program anything themselves. Moreover, we have ensured that it will be easy to extend its capabilities in the future and that it can function properly on most displays. This chapter outlines our approach towards creating the toolbox to ensure the realization of these goals.

3.1 BACKGROUND INFORMATION

We created our toolbox using the Unity[1] engine. As a game engine, Unity provides us with a solid foundation of functionalities and stock assets to build our Graphical User Interface (GUI) with, which saved us a lot of time and effort and allowed us to focus more on the design of the toolbox itself. Another benefit of Unity is that it supports building applications for many different platforms, and most of its assets are platform-independent, which furthers our goal of making the toolbox as hardware-independent as possible. In Unity, a program consists of one or more scenes, which are self-contained units consisting a tree of game objects and components. We use these scenes to separate the different parts of our application, which allows for a modular approach where adding a new functionality to the toolbox can be as simple as creating a new scene for it and adding it to the build settings. In our toolbox, scenes generally come coupled with one or more scripts that define behaviour specific to that scene. See Figure 3.1 for an overview of the scenes that make up our toolbox, and how they connect.

3.2 PERFORMING EXPERIMENTS

To allow custom experiments without any programming, our toolbox has a predefined set of functionalities, which the user can customize by providing two files. This section explains the roles of these files in the usage of our toolbox.

3.2.1 *The config file*

The user must provide a config file to customize the program's behaviour. We chose to use the JavaScript Object Notation (JSON) format

1 https://unity.com

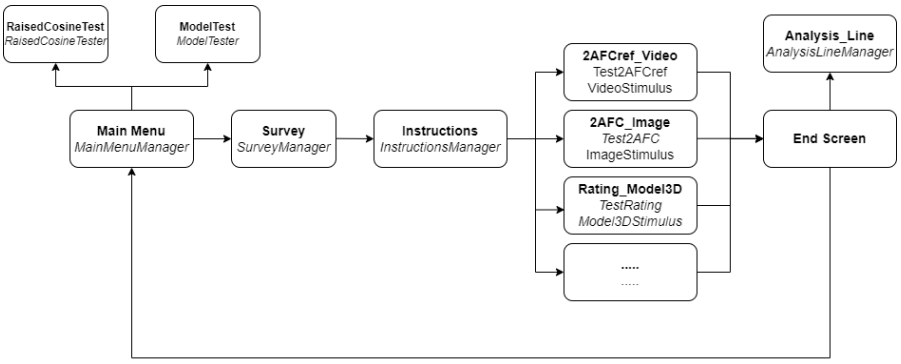


Figure 3.1: This diagram contains the scenes that make up our toolbox. The bold text shows the scene name, and the italic text below shows the script(s) associated with the scene. The arrows indicate the possible scene transitions. The Main Menu is the origin scene, which opens when the application starts.

because it is simple to understand and commonly used, so many users might already have experience with it. In C#, we can parse JSON files using the Newtonsoft JSON framework[2]. A complete list of the available parameters and their formats is given in Appendix A. The user is required to supply at least the three mandatory parameters:

1. Path: the absolute or relative file path to a folder containing the stimuli to display.
2. Experiment Method: the experiment method that the user intends to use.
3. Stimulus Type: the type of stimulus that the user intends to display during the experiment.

3.2.2 *The experiment specification*

The experiment specification contains crucial information about the trials that are performed during the experiment, such as the filenames of the stimuli. It is a space-separated text file in which every line represents one trial. It starts with a (possibly empty) set of warm-up trials, separated from the main trials by an empty line. The format of the lines depends on the experiment method, but it will generally require at least a set of filenames separated by spaces.

3.3 MODULARITY OF STIMULI AND EXPERIMENT METHODS

One of our key objectives was to ensure that stimuli and experiment methods were modular, and that new ones could easily be added

2 https://www.newtonsoft.com/json

in the future. This section explains our design for the stimuli and experiment methods, and the required steps to add a new stimulus or experiment method.

3.3.1 *Modularity of experiment methods*

In our toolbox, each experiment method has its own Unity scenes, generally requiring one scene per stimulus. We use an abstract class `TestManager` to define behaviour that should be common to all experiment methods. This behaviour includes initializing the trial scene, shuffling the trial order and saving its mapping, and progressing through the trials. For every experiment method, we create a class deriving from `TestManager` to define behaviour specific to the experiment method. This behaviour may include loading a new set of stimuli and storing the results of the trials. Creating a new experiment method thus requires creating a new set of scenes and a new `TestManager` sub-class.

3.3.2 *Modularity of stimuli*

As mentioned in Section 3.3.1, each experiment method has one Unity scene per stimulus. In other words, every stimulus has one Unity scene for each experiment method. We use an interface (`IStimulus`) as an abstraction for the stimuli. We opted to use an interface instead of an abstract class because the implementations of the stimuli are largely unrelated. Therefore, no common behaviour exists between them that could be provided by an abstract class. By using the interface, the `TestManager` and its sub-classes can function independently of the stimuli, because we define all stimulus-specific behaviour in the `IStimulus` implementation. This stimulus-specific behaviour includes loading the stimulus from a file and ensuring it displays correctly. Creating a new stimulus type thus requires creating a new set of scenes and a new `IStimulus` implementation.

3.4 GENERAL SCENES

This section provides an overview of the scenes in our toolbox that can appear regardless of the configured experiment method and stimulus.

3.4.1 *Main menu*

The main menu scene initially greets the user when launching the toolbox. As shown in Figure 3.2, the main menu displays the contents of both the config file and the experiment specification. Additionally, it allows the user to refresh the current files or select different ones.

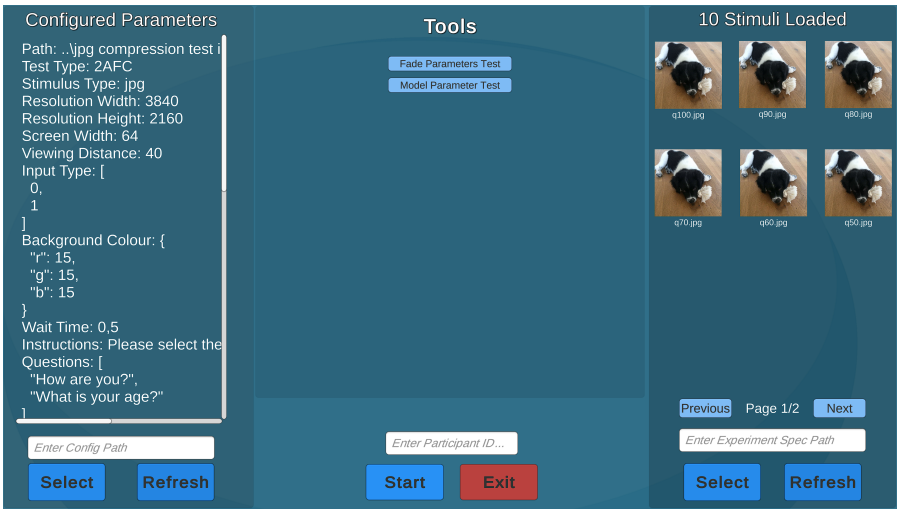


Figure 3.2: An example of what the main menu screen may look like. On the left, an overview of the config file is shown. On the right, an overview of the experiment specification is shown.

3.4.2 *Survey & instructions*

The survey and instructions scenes are both optional scenes that may appear depending on the *Survey* and *Instructions* configuration parameters. The survey presents the participant with a sequential list of questions, to which the participant may provide an answer by typing it into a textbox. The instructions scene displays user-defined instructions to the participant.

3.4.3 *End screen*

The end screen is the scene displayed after a participant finishes all the trials in the experiment. It informs the participant that the trial phase has concluded. Aside from this, it contains three buttons, which allow the user to close the toolbox, return to the main menu, or view a summary of the experiment that was just performed.

3.4.4 *Tools*

The toolbox contains two scenes that we categorize as tools. These are not used when performing experiments, but may be helpful when setting them up. They allow the user to test with different values for certain configuration parameters. The toolbox currently contains a tool for fade parameters (explained in Section 4.4.2) and a tool for model parameters (explained in Section 4.6).

4

EXPERIMENT METHODS & STIMULI

Our toolbox comes built-in with three experiment methods and three stimuli types. Aside from these, 20 optional configuration parameters are available to the users. This chapter describes these experiment methods, stimuli, and configuration parameters, as well as their implementation.

4.1 EXPERIMENT METHOD: 2AFC

The 2AFC task presents the participant with concurrent pairs of stimuli. The participant must then choose one of the stimuli based on some property. Figure 4.1 displays a sample trial.

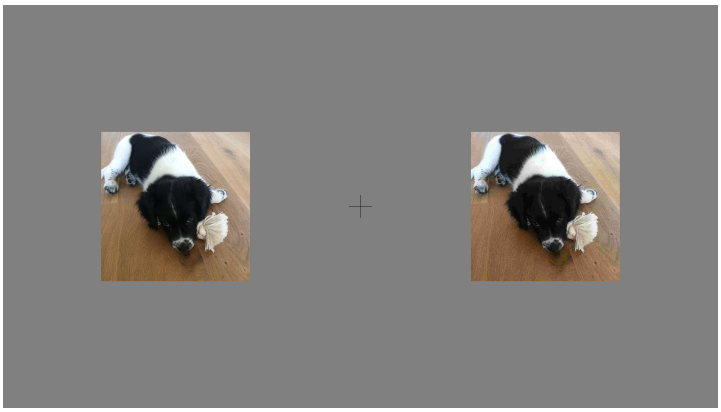


Figure 4.1: An example of what a 2AFC trial might look like when running an experiment with our toolbox.

4.1.1 *Eccentricity*

As seen in Figure 4.1, the two stimuli are presented at an equal distance from the centre of the screen. The length of this distance is determined by the *eccentricity*, which the user must provide for each trial in the experiment specification. The eccentricity is the horizontal angle between the centre of the screen and the centres of the two stimuli from the participant's perspective. In the 2AFC scenes, the stimuli are anchored to the centre, so we need to calculate the distance from the centre in pixels corresponding to the given eccentricity. We can compute this pixel distance in two ways depending on the parameters provided.

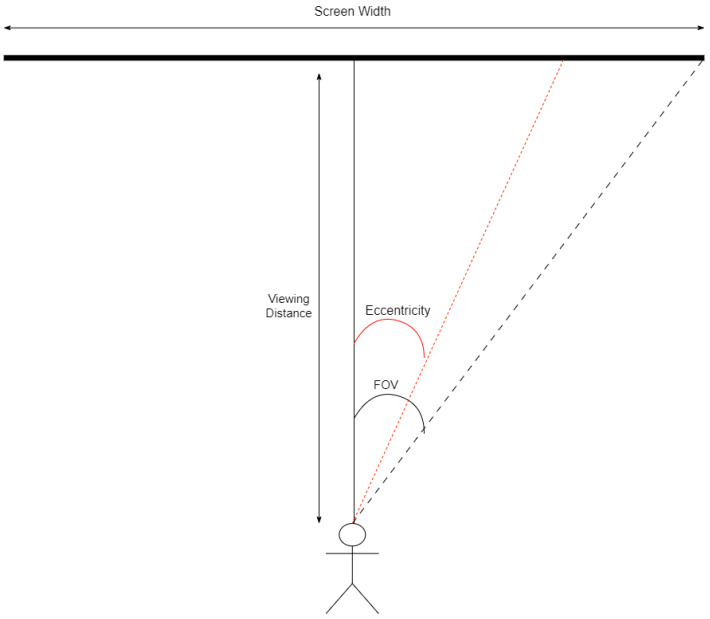


Figure 4.2: This diagram illustrates the configuration parameters related to the eccentricity calculation, as seen from above. The thick horizontal line represents the user's screen, and the stick-figure represents the user.

4.1.1.1 *Tangent*

The first approach requires the user to supply the *Screen Width* and *Viewing Distance* parameters (both in centimetres). Additionally, the user may provide the *Resolution Width* parameter[1]. As seen in Figure 4.2, there is a right triangle where the eccentricity is the angle and the viewing distance is the adjacent side. Using simple trigonometry, we can find the desired pixel distance d:

$$d = \frac{\mathit{Viewing\ Distance} \cdot \tan(\mathit{eccentricity})}{\mathit{Screen\ Width}} \cdot \mathit{Resolution\ Width}$$

4.1.1.2 *Field of view*

The second approach requires the user to supply the *Field Of View* parameter, which contains the angle between the centre and edge of the screen from the user's perspective, as seen in Figure 4.2. The calculation is then:

$$d = \frac{\mathit{eccentricity} \cdot \mathit{Resolution\ Width}}{2 \cdot \mathit{Field\ Of\ View}}$$

The advantages of the second approach are that the calculation is simple and works for curved displays. However, the field of view

1 The *Resolution Width* parameter specifies the resolution width of the display in pixels, which will default to the resolution width of the toolbox window if omitted.

might be difficult to measure and require multiple measurements for different viewing distances.

4.1.2 *Target*

The 2AFC task requires the participant to stare at the centre of the screen, at which a target is displayed. As shown in Figure 4.3, the target appears as a cross and is surrounded by a mask that bounds its display size. Note that the real length of the cross' lines is much larger than shown here, extending far beyond the mask.

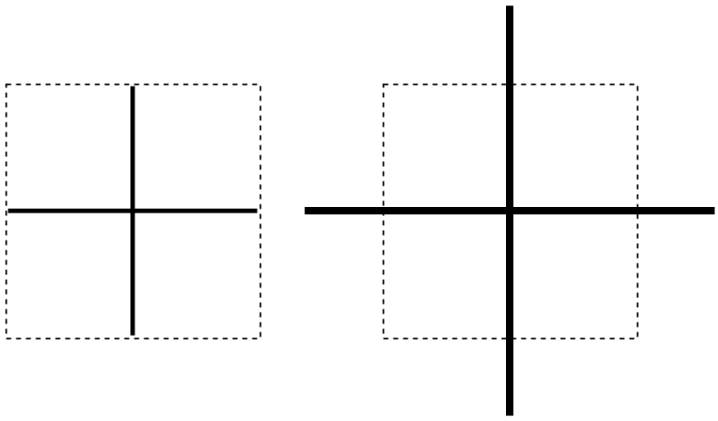


Figure 4.3: The target that appears at the centre of the screen during the 2AFC trials. The target is placed inside a mask, indicated by a dashed line. Any portion of the target that is outside of the mask will not be rendered. On the right, the scale of the target was increased, but the scale of the mask remained the same. As a result, the target appears to be thicker.

The user can modify the size of the cross through the *Target Scale* configuration parameter. Doing this will alter the scale of the mask, meaning that either more or less of the target is shown. The user can also modify the thickness of the cross through the *Target Thickness* configuration parameter. Doing this will alter the scale of the target image. Since only the scale of the target image changes, and not that of the mask, the image will appear thicker or thinner without its dimensions changing.

4.2 EXPERIMENT METHOD: 2AFC WITH REFERENCE

The Two-Alternative Forced Choice with reference (2AFCref) method is similar to the 2AFC method but contains three stimuli instead of two. The third stimulus is used as a reference and is displayed in the centre of the screen, replacing the target in the 2AFC method, as seen in Figure 4.4.

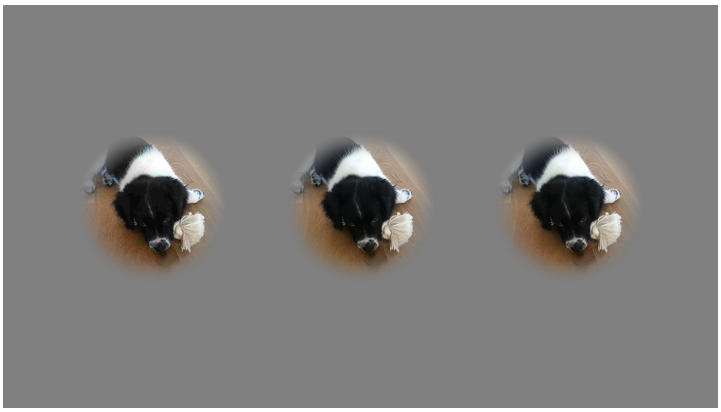
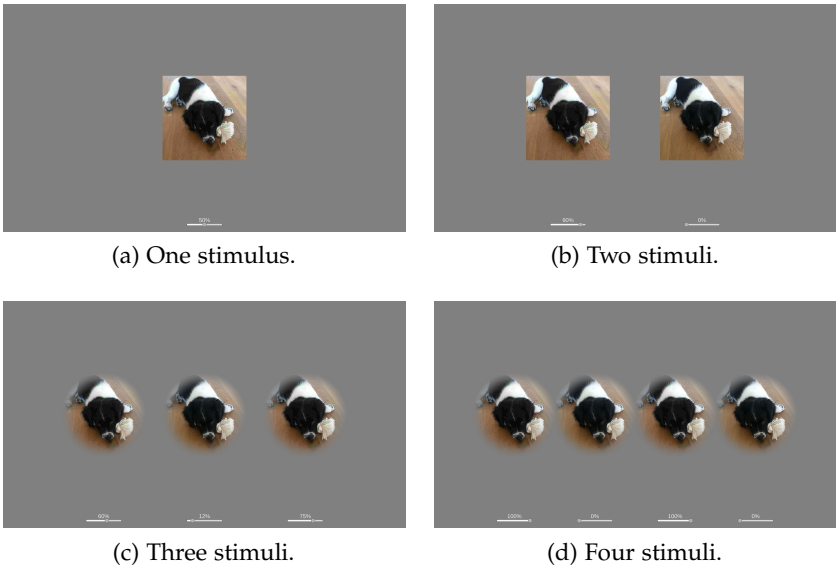


Figure 4.4: An example of what a 2AFCref trial might look like when running an experiment with our toolbox. The image in the middle is the reference stimulus.

4.3 EXPERIMENT METHOD: RATING

The rating method presents the participant with concurrent sets of stimuli. The size of the set is arbitrary but remains constant throughout the experiment. Based on some property, the participant must assign a score to every stimulus. The stimuli will have an equal amount of space between their centres. As seen in Figure 4.5, our implementation includes a set of sliders, which the participant may use to give their rating for each stimulus.



(a) One stimulus.

(b) Two stimuli.

(c) Three stimuli.

(d) Four stimuli.

Figure 4.5: Examples of what a rating trial may look like with one, two, three, or four stimuli. Every stimulus is accompanied by a slider that the participant can set to a value from 0% to 100%.

In the unity scenes for the rating test, we implemented the base case where only one stimulus is present. Based on the number of stimuli

required, this base stimulus is duplicated, along with its slider. Then, the stimuli and sliders are moved horizontally to achieve the equal spacing.

4.4 STIMULUS TYPE: IMAGE

The first and simplest stimulus type we added to the toolbox was the image stimulus. It displays an image stored using either the PNG or JPEG file format. The images are displayed in their true size, meaning that one pixel on the display exactly matches one pixel on the image. This approach prevents any bias caused by upsampling or downsampling of the pixel values that occur during the scaling of images. We implemented the image stimulus by using Unity's built-in UI `Image`[2]. This approach allows us to read a stored image into a 2D texture. Depending on the configuration parameters, we then apply some modifications to the texture, such as the screen calibration (Section 4.4.1) and blending (Section 4.4.2). Finally, the texture is converted to a sprite and loaded into the image, which we then resize to match the sprite's dimensions. Since the image is a UI component, it must be placed inside a Unity `Canvas`[3].

4.4.1 *Screen calibration*

The user may adjust the RGB pixel values of the displayed images by providing the *Screen Calibration* configuration parameter. It consists of four floating point numbers a, b, c, γ for each colour channel, which are used according to the following formula:

$$y = a + (b + cx)^{\gamma},$$

where x is the old value of the colour channel, and y is the new value. This calculation is then applied three times to each pixel, once for every colour channel (red, green, and blue). Note that the 0-255 range is used for the RGB values. By using the screen calibration, the user can achieve a near-identical look for the same stimuli on displays with varying brightness levels, contrasts, and colour settings. For example, the user may set $a = 10$ for each colour channel, which causes the image to have a slightly increased brightness.

2 https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Image.html

3 https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html

4.4.2 *Blending images*

The user may want to blend images into the background, to reduce the sharpness of their edges. Our toolbox provides the user with this functionality through raised cosine filtering:

$$H(f) = \begin{cases} 1 & \text{if } |f| \leq \frac{1-\beta}{2T} \\ \frac{1}{2}[1 + \cos(\frac{\pi T}{\beta}[|f| - \frac{1-\beta}{2T}])] & \text{if } \frac{1-\beta}{2T} < |f| \leq \frac{1+\beta}{2T} \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

The formula shown in Equation 4.1 is applied to the alpha colour channel of the pixels, which represents the transparency. Note that $0 \leq \alpha \leq 1$. The value of f is calculated for each pixel based on its position in the image. We use parameters u and v, which are defined as follows:

$$u = \frac{\textit{horizontal pixel index}}{\textit{image width}} - 0.5$$

$$v = \frac{\textit{vertical pixel index}}{\textit{image height}} - 0.5$$

The user can customize the behaviour of the blending through three different configuration parameters:

1. *Beta*: this parameter allows the user to choose the value of the β constant seen in Equation 4.1. The β constant controls how quickly the fade goes from $\alpha = 0$ to $\alpha = 1$.

2. T: this parameter allows the user to choose the value of the T constant seen in Equation 4.1. The T constant controls how far from the edge the fade starts, i.e. where α starts to become lower than 1.

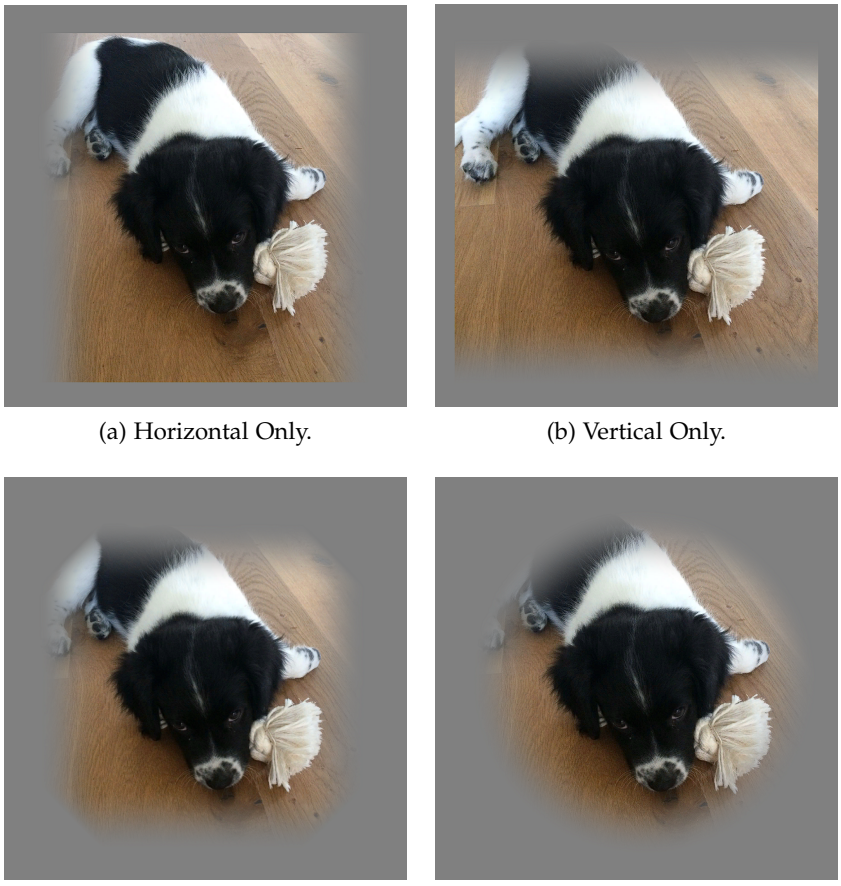
3. *Blend Mode*: through this parameter, the user may select the blending type to use. Our toolbox supports four different types of blending, which will be detailed below. Examples of each blending function can be seen in Figure 4.6.

HORIZONTAL ONLY This blending function will only blend the left and right sides of the image into the background. The value of α is calculated as $\alpha = H(u)$.

VERTICAL ONLY This blending function will only blend the top and bottom of the image into the background. The value of α is calculated as $\alpha = H(v)$.

NORMAL This blending function will combine the horizontal and vertical blending. The value of α is calculated as $\alpha = H(u) + H(v) - 1$.

CIRCULAR This blending function will circularly blend the image. The value of α is calculated as $\alpha = H(\sqrt{u^2 + v^2})$.



(a) Horizontal Only. (b) Vertical Only.

(c) Normal. (d) Circular.

Figure 4.6: Examples of the four blending functions our toolbox provides. All examples were made using $T = 1.30, \beta = 0.25$.

4.5 STIMULUS TYPE: VIDEO

The *video* stimulus displays a video that is stored in MP4 format. Like images, the videos are displayed at their true size. Obtaining the video's dimensions required the use of FFMPEG[4], which allows us to read the metadata of stored videos. We implemented videos by using Unity's built-in `VideoPlayer`[5] component. The video player also requires a render target, for which we chose to use a `RawImage`[6] component since we can easily resize it to match the video's original dimensions. Displaying a video during runtime is as simple as providing the video player with the video's Uniform Resource Locator (URL).

4 https://ffmpeg.org
5 https://docs.unity3d.com/Manual/class-VideoPlayer.html
6 https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-RawImage.html

Since the video player and raw image are both UI components, they must be inside a canvas. At the time of writing, the video player has only been tested on Windows 10 and may not function properly on other platforms due to compatibility issues.

4.5.1 *Blending videos*

Similarly to images, we can blend videos into the background. Unlike images, which are static, videos may change their pixels at every frame, making it inefficient to blend the videos by altering their pixel values. Instead, the video stimulus uses a mask image, which is entirely the same colour as the background and will have an inverted transparency compared to the image blending. We then use the same calculations as in Section 4.4.2, but invert the final value: $\alpha' = 1 - \alpha$. As a result, the mask becomes more opaque towards the edges, while being transparent near the centre.

4.6 STIMULUS TYPE: MODEL3D

The final stimulus we implemented is the *Model3D* stimulus, which can display 3D objects stored using the OBJ file format. Unlike images and videos, which use two-dimensional scenes, correctly displaying 3D objects requires three-dimensional scenes. This extra dimension introduces notable differences, which we discuss in Section 4.6.1 and Section 4.6.2. Another difference is that models require a light source, which we place at the same position as the camera (both are at the origin). Loading in new models during runtime is not easy to do in Unity, as Unity provides no way of doing this. Therefore, we used the Runtime OBJ Importer[7], which is a free asset from the Unity asset store that allows us to load in models from OBJ files during runtime. After loading a model, all we do is copy some properties from the previous model (such as its position and orientation), which we then remove from the scene.

4.6.1 *Projection*

Since models use three-dimensional scenes, a projection to 2D is needed. We chose to use an orthographic projection because it allows identical models to look the same, regardless of where they are placed on the screen. When using a perspective projection, models may look differently depending on their positions, even if they are identical, as shown in Figure 4.7. We set the size of the orthographic projection to

7 https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547

the resolution of the user's display, which allows us to use identical coordinates for models, images, and videos.

(a) Orthographic.

(b) Perspective.

Figure 4.7: The same models when using an orthographic projection and a perspective projection. The right model looks different when using a perspective projection.

4.6.2 *Transformations*

Since the models do not have any constraints regarding their display size, and due to their three-dimensional nature, we allow a higher degree of customization for how models should be displayed. This subsection will cover the configuration parameters exclusive to 3D models.

MODEL SCALE The *Model Scale* parameter allows the user to apply a three-dimensional non-uniform scaling transformation to the displayed models. Since 3D objects can vary in size, this parameter allows the user to shrink or enlarge their objects to fit properly on the screen.

MODEL ROTATION The *Model Rotation* parameter allows the user to apply a three-dimensional rotation transformation to the displayed models. Since 3D objects may appear differently depending on their rotation, this parameter allows the user to ensure that the rotation of their model is correct.

SYMMETRIC MODELS This parameter allows the user to toggle whether or not the displayed models should be horizontally symmetric, if possible. We do this by modifying the models on the right half of the screen. For these models, the x component of their scale and the y and z components of any rotations performed on them are made negative.

MODEL DISTANCE This parameter allows the user to change the distance between the models and the camera. Only the distance along the z-axis can be modified in this way. Since we use an orthographic projection, this distance will not affect the models' displayed sizes; only the lighting will change.

OTHER PARAMETERS Aside from the transformations, there are some other configuration parameters that the user may provide to customize the display of the models. These parameters are listed below:

- *Model Colour*: allows the user to set the colour of the model.
- *Glossiness*: allows the user to determine how glossy/reflective the model should be.
- *Lighting Type*: allows the user to choose the type of light source to use. Our toolbox currently supports ambient, point, spot, and directional light sources.
- *Lighting Intensity*: allows the user to adjust the intensity of the light source.

4.7 GENERAL CONFIGURATION PARAMETERS

The previous six sections already discussed some configuration parameters that were specific to some experiment method(s) or stimulus type(s). This subsection will cover the parameters that can be used with any combination of experiment method and stimulus type.

4.7.1 *Background Colour*

The *Background Colour* parameter allows the user to specify, in RGB values, the colour of the background during trials. This parameter can be important for some experiments, since the background colour may affect the participant's colour perception. For example, a bright background might cause the user's vision to adapt to higher brightness, making it more difficult to differentiate between darker colours in a stimulus.

4.7.2 *Wait Time*

The *Wait Time* parameter allows the user to specify the number of seconds it should take for the next set of stimuli to appear on the screen after finishing the current trial. During this wait time, the user's progress will appear in the centre of the screen. This wait time may also reduce bias created by the user's eyes adapting to the colour of a stimulus.

5 RESULTS & ANALYSIS

When performing psychovisual experiments, it is crucial to store the results and to ensure that they are easy to analyse. Our toolbox provides users with a simple way to store, distinguish, and analyse their results. This chapter explains our design for this system and shows the results of a JPEG compression experiment we performed using the toolbox.

5.1 STORED INFORMATION

When an experiment is performed, we store the following types of information:

- *Test ID*: an identifier that is assigned to the experiment by the user. It is requested from the user on the main menu before starting the experiment.
- *Survey*: a list of question-answer pairs containing the questions that were asked during the survey and the responses provided by the participant.
- *Test Type*: the type of test that was performed, which is a combination of the stimulus type and experiment method (for example: 2AFC_Image).
- *Mapping*: the mapping that was applied to the list of trials to randomize the order.
- *Results*: a list of trial-response pairs containing information about the trial presented and the participant's response. The format of this data depends on the experiment method.

5.2 STORAGE METHOD

The results of an experiment are stored in two or three different text files, depending on the user's preferences.

5.2.1 *Human-readable log*

The human-readable log is designed in a way that makes it easy for humans to understand. It contains all the types of information listed in Section 5.1. An example of what this log file might look like is shown in Figure 5.1a. The mapping a -> b means that the a^{th} trial

in the experiment specification was shown as the b^{th} trial during the experiment.

```
Participant id: 123

Question 1: What is your name?
Answer: Gerben

Question 2: What is your age?
Answer: 21

Test type is 2AFC_Image

Mapping:
0 -> 7, 1 -> 6, 2 -> 5, 3 -> 3, 4 -> 2, 5 -> 0, 6 -> 4, 7 -> 1, 8 ->
8,

Trial 0: On the left: q40.jpg, on the right: q100.jpg, with e=20
Participant chose the stimulus on the right
Trial 1: On the left: q20.jpg, on the right: q100.jpg, with e=20
Participant chose the stimulus on the left
Trial 2: On the left: q50.jpg, on the right: q100.jpg, with e=20
Participant chose the stimulus on the right
Trial 3: On the left: q60.jpg, on the right: q100.jpg, with e=20
Participant chose the stimulus on the left
Trial 4: On the left: q30.jpg, on the right: q100.jpg, with e=20
Participant chose the stimulus on the right
Trial 5: On the left: q100.jpg, on the right: q70.jpg, with e=20
Participant chose the stimulus on the left
Trial 6: On the left: q100.jpg, on the right: q80.jpg, with e=20
Participant chose the stimulus on the right
Trial 7: On the left: q100.jpg, on the right: q90.jpg, with e=20
Participant chose the stimulus on the left
Trial 8: On the left: q10.jpg, on the right: q100.jpg, with e=20
Participant chose the stimulus on the right
```

(a) The human-readable log

```
123
2AFC_Image
9
0:7 1:6 2:5 3:3 4:2 5:0 6:4 7:1 8:8
q40.jpg q100.jpg 20 1
1
q20.jpg q100.jpg 20 1
0
q50.jpg q100.jpg 20 1
1
q60.jpg q100.jpg 20 1
0
q30.jpg q100.jpg 20 1
1
q100.jpg q70.jpg 20 0
0
q100.jpg q80.jpg 20 0
1
q100.jpg q90.jpg 20 0
0
q10.jpg q100.jpg 20 1
1
```

(b) The machine-readable log

Figure 5.1: An example of what the log files might look like. Both logs were generated by the same experiment

5.2.2 *Machine-readable log*

This log is designed to be easy for a program to parse. The first three lines contain the *Test ID*, *Experiment Type*, and the number of trials in the experiment. The fourth line contains the mapping, which follows the same logic as the one in the human-readable log. After that, the *Results* are displayed on separate pairs of lines. This log does not include the *Survey*.

5.2.3 *Warm-up log*

This log contains the *Results* for the warm-up trials, and will only be created if the user has provided the *Log Warmups* configuration parameter with a value of `true`. The format of the *Results* list is identical to that of the human-readable log.

5.3 EXPERIMENT SUMMARY

Our toolbox provides a single tool that is used to provide a summary of the results of an experiment. It only works for the 2AFC and 2AFCref experiment methods. It will display a list of the performed trials, some helpful information, and the user's response. It obtains its data by parsing the machine-readable log. As seen in Figure 5.2, the tool shows the stimuli for each trial, with the chosen stimulus surrounded by a

red border. The list of trials spans multiple pages, with up to four trials on each page.

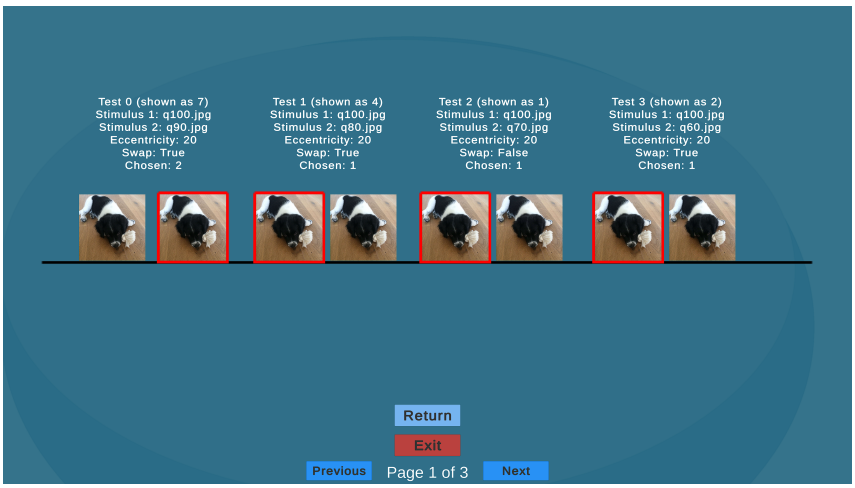


Figure 5.2: An example of the analysis tool.

5.4 VALIDATION WITH A JPEG COMPRESSION EXPERIMENT

In order to test the usability of our toolbox for psychovisual experiments, we have performed our own experiment. In this experiment, we investigated the visibility of JPEG compression artifacts across different compression levels, at varying eccentricities.

5.4.1 *Experiment set-up*

For this experiment, we generated stimuli with different JPEG compression levels using the MATLAB `imwrite` function. For JPEG images, the `imwrite` function allows for the specification of the `Quality` argument, which we used to control the compression level. We chose the following quality levels: $q \in \{1, 2, 4, 8, 16, 20, 50, 75, 100\}$. We showed every quality level in a 2AFC trial, where the other stimulus was always the image with quality $q = 100$. These trials were performed once for each of the following eccentricities: $e \in \{5, 10, 20, 35, 55\}$. We performed this experiment 6 times to measure the detection rate, i.e. in how many of the experiments the most compressed stimulus was correctly identified. All repetitions of this experiment were performed by the same person. Additional information about the experiment is shown in Table 5.1

PARAMETER	VALUE
Viewing distance (cm)	60
Display FOV	114.5916°
Display peak luminance (cd/m^2)	158.9
Display colour space	sRGB
Background colour (RGB)	[186, 186, 186]
Background luminance (cd/m^2)	78.1
Image dimensions (px)	256x256

Table 5.1: This table contains parameters for the experiment we performed.

5.4.2 *Experiment results*

To analyze the results of this specific experiment, we created a script that parses all of the machine-readable logs. After parsing, the Scottplot[1] library was used to create a plot that displays the results. This plot can be seen in Figure 5.3. We opted to plot the quality (x-axis) against the detection rate (y-axis) for each eccentricity, because it allows us to distinguish the results for different eccentricities and quality levels. It is clear from Figure 5.3 that the detection rate of low-quality

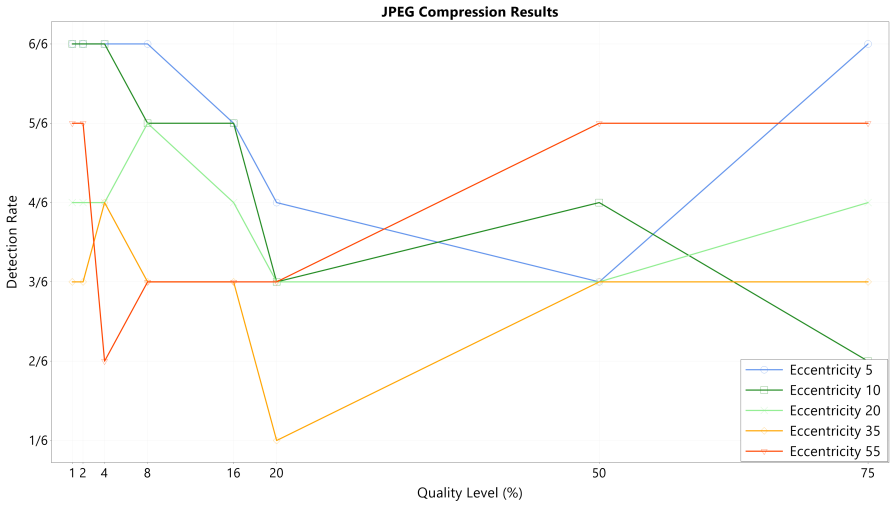


Figure 5.3: A graph displaying the detection rate (y-axis) for different combinations of compression levels (x-axis) and eccentricities.

stimuli was higher at lower eccentricities, while the detection rate of high-quality stimuli is mostly unaffected by the eccentricity. When the difference between two stimuli was not visible, there was a 50% chance to choose the right stimulus. Thus the expected detection rate for such stimuli would be 3/6. Since only six experiments were performed,

1 https://scottplot.net

outliers (such as $q = 75, e = 55$) are expected to be very common. Therefore, the data at higher quality levels and eccentricities is not accurate enough to draw conclusions from.

6 VIRTUAL REALITY

We have created an adaptation of our toolbox that performs adequately on VR displays, specifically the Oculus Rift. This chapter explains the changes we had to make to our toolbox so that it would work on the Oculus Rift.

6.1 UNITY SET-UP

To develop the application, we first had to install the Oculus XR-Plugin. We also chose to use Unity's XR Interaction Toolkit[1], which provides many different stock assets. These assets allowed us to focus more on the development of the toolbox itself instead of having to implement basic functions, such as the behaviour of the controllers. We chose to use the `Complete XR Origin Set Up` prefab asset, which is a set of GameObjects and Components that work together to transform data from the Extended Reality (XR) tracking subsystems into world space [11].

6.2 SCENES

For the VR adaptation, we had to modify all of our scenes. This section shows some common updates we had to make to all scenes.

6.2.1 *VR Foundation*

The `VR Foundation` is a prefab asset we created that contains all the standard components needed for a VR scene. These components include a floor for the user to stand on, a light source to illuminate the user's surroundings, and a `Complete XR Origin Set Up`. We then added this asset to each scene in our toolbox.

6.2.2 *Canvas*

In Unity, all UI components must be placed inside a `Canvas` to be rendered. For the original toolbox, all our scenes used canvases set as screen overlays. For XR scenes, this is not an option. Instead, the canvas must be placed in world space. The canvases now also require a `Tracked Device Graphic Raycaster` component to handle interac-

1 https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/index.html

tions with its UI components, such as clicking buttons. We thus had to modify every single canvas in every scene.

6.2.3 *Projection*

The three-dimensional scenes in the original toolbox used an orthographic projection since it allowed identical models to look identical regardless of their positions in the scene. VR scenes, however, do not support orthographic projection. Therefore, we had to change some cameras to use a perspective projection. As a result, identical models may not look the same.

6.3 ECCENTRICITY

Since we now use three-dimensional scenes where the user may look around by moving their head, we had to ensure that the stimuli stay at the correct eccentricity while still being displayed correctly. To accomplish this, we created a hierarchy of game objects, which is shown in Figure 6.1. The solution can be interpreted as having the stimuli on the ends of straight line segments. As the user looks around, these line segments move and rotate to face the user. Additionally, the stimuli themselves are rotated to face the user. We calculate the positions of points 2 by applying a rotation of α to point 1.

6.4 INPUT

The Oculus Rift uses controllers for its input, meaning we had to completely overhaul our input systems, which relied on keyboard and mouse inputs.

6.4.1 *Mouse clicks*

For mouse click interactions, we only had to replace the `Graphic Raycaster` components of the canvases with `Tracked Device Graphic Raycaster` components. Then, the clicking interaction can be performed by pointing at the interactable object with one of the controllers and pressing its trigger.

6.4.2 *Keyboard input*

For the trial scenes, we had to replace all of the keyboard button inputs with inputs of the controller buttons. We implemented this by checking whether any buttons are pressed at every frame update. For the 2AFC and 2AFCref trials, we allow the user to click either the primary (A/X), secondary (B/Y), or grip button on either controller to choose

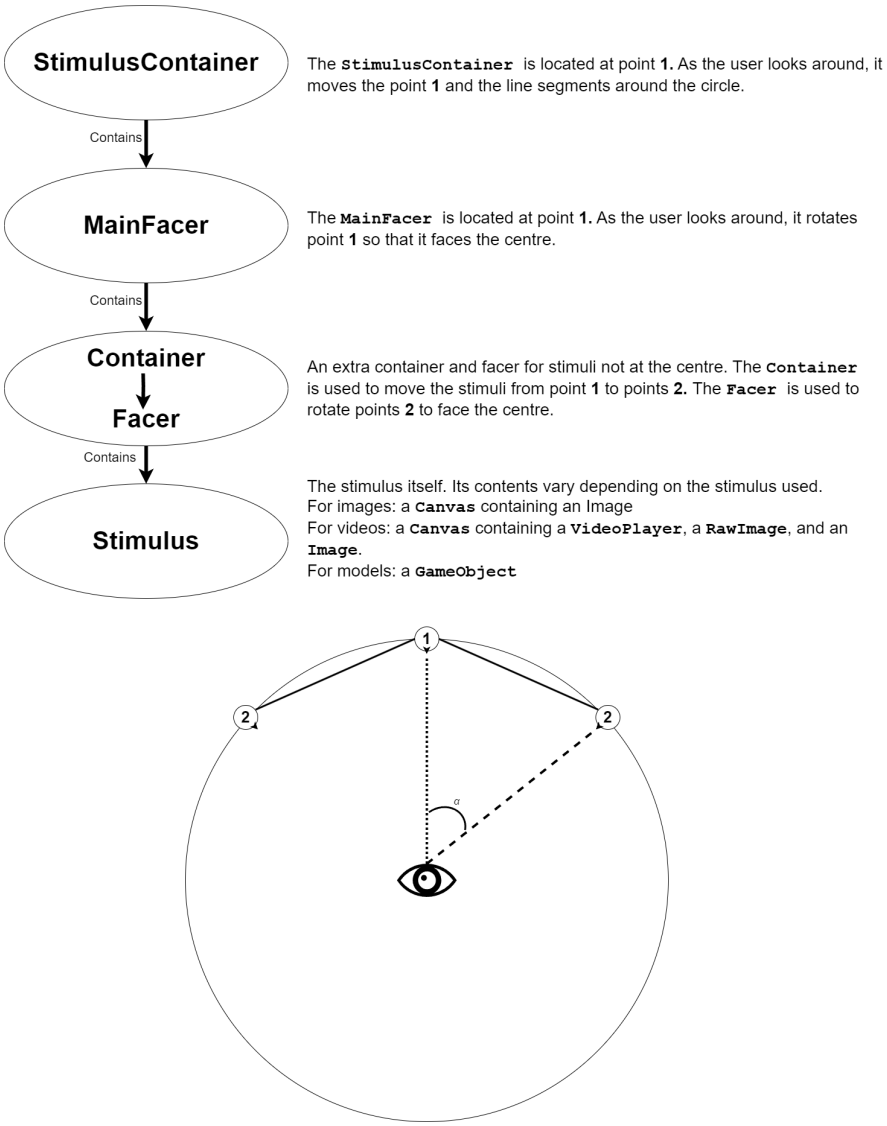


Figure 6.1: Our hierarchy of game objects that are used together to make stimuli appear at the correct angle and position.

a stimulus. Using the left controller indicates picking the left stimulus and vice versa. For the rating trials, the user must simultaneously click the primary buttons of both controllers to confirm their ratings.

6.4.3 *Virtual keyboard*

While we managed to replace the keyboard inputs with controller inputs for the trial scenes, we still needed some way for the user to enter strings, e.g. to answer the survey questions. To this end, we implemented a virtual keyboard, seen in Figure 6.2, which appears when the user selects a text input field. The user can use the controllers to click on the keyboard's buttons, which will add the character corresponding to the chosen key to the input field.

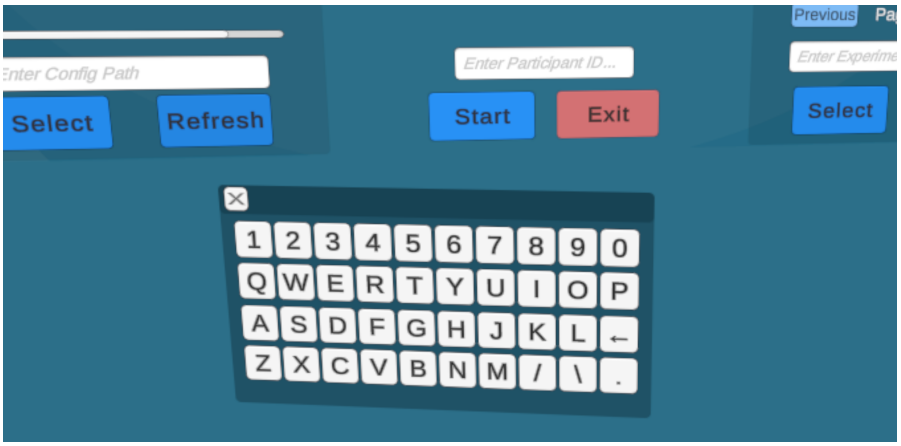


Figure 6.2: The virtual keyboard we created for the VR adaptation of our toolbox.

6.5 EVALUATION

As explained in the previous sections, we have successfully adapted all of the core functionalities of our toolbox to work on the Oculus Rift. All of the stimuli and experiment methods that we implemented for the original toolbox can also be used in VR. We have reworked our user input systems to be compatible with the Oculus Rift controllers. While largely successful, our adaptation is not perfect; there are minor differences with the original toolbox, such as the projection mentioned in Section 6.2.3.

7 CONCLUSIONS

Our project's goal was to create a toolbox that could be used to perform psychovisual experiments without any programming and to which new stimuli types and experiment methods could both easily be added. Other important objectives included making our toolbox display-independent and creating a VR adaptation.

By using a configuration file, our toolbox can perform different experiments without any programming efforts from the user. Making the stimuli and experiment methods modular was as simple as using an abstract class and interface to define behaviour specific to experiment methods and stimuli and creating different Unity scenes for each combination.

Display-independence has been partially achieved through some configuration parameters, which provide functionalities such as gamma correction. However, some of these could not be implemented for all stimuli. Gamma correction, for instance, currently only supports image stimuli.

We adjusted all of the scenes in the original toolbox to function adequately on VR displays. There are however some functionalities not supported in Unity VR, such as the orthographic projection originally used for model stimuli. Nevertheless, we consider the adaptation to be successful.

By performing our own JPEG compression experiment with the toolbox, we have verified that we can use the toolbox to conduct psychovisual experiments without requiring any programming. Therefore, we conclude that we have successfully designed a user-friendly desktop application for conducting psychovisual experiments.

8 FUTURE WORK

While we consider the project a success, there are many ways in which the project could be improved in the future. The most notable ones are listed below:

- *Platform independence*: We mainly created the project to work on devices using Windows 10 as their operating system. Since Unity offers good cross-platform support, making the toolbox work on other platforms should be possible. However, fundamental differences in the hardware and deployment methods mean that parts of a project may not port between platforms without change.[1] For example, we ran into an issue where the Unity VideoPlayer component would not be able to load videos during runtime on a Macbook running OSX. This problem was the only one we encountered, but other platforms like Linux and Android have not been tested.

- *Display independence*: While we implemented some ways to make our toolbox more display independent, not all of these ways support all of our stimulus types. Moreover, other parameters could be implemented to improve display independence, such as adjusting the frame rate of videos. Future work should include improving the current parameters and adding new ones.

- *Improve stimuli support*: Currently, our toolbox supports only a small number of file types for each stimulus type. For example, 3D models only support the OBJ file format. Many other popular file formats exist for each stimulus type (FBX and STL for 3D models, for example). In the future, we should extend the toolbox to support these file formats, which we can do either by improving the existing stimulus implementations or adding new ones.

- *Improving the VR adaptation*: Some functionalities could not be adapted to VR, including the orthographic projection and making images and videos pixel-perfect. Moreover, our VR adaptation has only been tested on the Oculus Rift. Many other VR headsets exist, which may be made by different companies, or have different specifications (e.g. resolution). In the future, our VR adaptation should be improved and made to work on other popular VR headsets, such as the Oculus Quest and Microsoft Hololens.

1 https://docs.unity3d.com/Manual/CrossPlatformConsiderations.html

- *Configuration file builder*: Our toolbox uses configuration files in JSON format. While the JSON format is easy to understand, it may still take time to get used to working with it. Adding a tool that provides a GUI in which users can create a configuration file without requiring knowledge of the JSON format would increase the user-friendliness of our toolbox.
- *Miscellaneous quality of life improvements*: Many small improvements should be made to our toolbox to improve the user experience. Some examples are the following:
  - A file selector may be added so that users do not have to determine the path to a file or directory.
  - A way to use the experiment summary with pre-existing log files may be added.
  - More experiment summary types, such as one supporting the rating experiment, may be added.
  - Additional tools that may be used to test the values for specific configuration parameters, such as the screen calibration, may be added.

A

CONFIGURATION PARAMETERS

This appendix lists all of the configuration parameters that are available to the user, including their required format and a brief description of their function.

A.1 MANDATORY PARAMETERS

This section includes the parameters that must be included in every configuration file

PATH (*string*)
The file path to a folder containing the stimuli to use for the experiment.

TEST TYPE (*string*)
The experiment method that should be used.

STIMULUS TYPE (*string*)
The file extension of the stimuli that should be used.

A.2 ECCENTRICITY PARAMETERS

This section includes the parameters that are used for the eccentricity calculation of the 2AFC and 2AFCref experiment methods. When using this experiment methods, it is mandatory to include either the `Field Of View` parameter, or the `Viewing Distance` and `Screen Width` parameters. When using the VR adaptation of the toolbox, these parameters will be ignored.

FIELD OF VIEW (*float*)
The angle (in degrees) between the centre of the screen and edge of the screen as measured from the viewing point.

VIEWING DISTANCE (*float*)
The distance (in centimetres) between the viewing point and the centre of the screen.

SCREEN WIDTH (*float*)
The distance (in centimetres) between the edges of the display.

A.3 GENERAL OPTIONAL PARAMETERS

This section includes the parameters that are optional, but may be used for any combination of experiment method and stimulus types. Each one of these has a listed default value.

RESOLUTION WIDTH (*integer*, default: read from device)
The width-component of the device's resolution in pixels.

RESOLUTION HEIGHT (*integer*, default: read from device)
The height-component of the device's resolution in pixels.

BACKGROUND COLOUR (*colour*, default: neutral gray)
The colour (in RGB values) that the background should be during trials.

WAIT TIME (*float*, default: `1.0`)
The amount of time (in seconds) between trials during which the progress counter should be displayed.

INSTRUCTIONS (*string*, default: ϵ)
The instructions that should be shown to the user before starting the experiment.

QUESTIONS (*string array*, default: `[]`)
The questions that should be asked of the user before starting the experiment.

LOG WARMUPS (*boolean*, default: `false`)
Whether or not the results of the warm-up trials should be recorded.

A.4 SPECIFIC OPTIONAL PARAMETERS

This section includes the parameters that are optional, and may only be used for a subset of the combinations of available stimulus types and experiment methods.

INPUT TYPE (*2AFC & 2AFCref*, *integer array*, default: `[0]`])
The types of input that may be used to select stimuli during the trials. The two currently available types are left/right arrow keys (`0`) and mouse clicks (`1`).

TARGET THICKNESS (*2AFC*, *float*, default: `1.0`)
A multiplier for the thickness of the target that appears at the centre of the screen.

TARGET SCALE (*2AFC*, *float*, default: `1.0`)
A multiplier for the size of the target that appears at the centre of the screen.

SCREEN CALIBRATION (*Image*, *object*, default: identity calibration)
This parameter contains the constants `a`, `b`, `c`, and `gamma` for each colour channel. These are used for the screen calibration functionality discussed in Section 4.4.1.

BLEND MODE (*Image & Video*, *string*, default: `"Normal"`)
Which type of blending should be used for the blending functionality discussed in Section 4.4.2.

BETA (*Image & Video*, *float*, default: `0.0`)
The value of the β constant used for the blending functionality discussed in Section 4.4.2.

T (*Image & Video*, *float*, default: `0.0`)
The value of the T constant used for the blending functionality discussed in Section 4.4.2.

SYMMETRIC MODELS (*3D Model*, *boolean*, default: `true`)
Whether or not the displayed models should be symmetric.

MODEL SCALE (*3D Model*, *3D vector*, default: identity scale)
A non-uniform scaling transformation that should be applied to the displayed models.

MODEL ROTATION (*3D Model*, *3D vector*, default: identity rotation)
A three-dimensional rotation transformation that should be applied to the displayed models.

MODEL COLOUR (*3D Model*, *colour*, default: white)
The colour that should be applied to the displayed models.

GLOSSINESS (*3D Model*, *float*, default: `0.5`)
How glossy the displayed models should look.

LIGHTING TYPE (*3D Model*, *string*, default: `"Ambient Only"`)
The type of light source that should be used in the trial scene.

LIGHTING INTENSITY (*3D Model*, *float*, default: `1.0`)
How luminous the light source in the trial scene should be.

BIBLIOGRAPHY

[1] *American Psychology Association Dictionary of Psychology*. Accessed: 20-02-2023. URL: https://dictionary.apa.org/psychophysics.

[2] George A. Gescheider. *Psychophysics: The Fundamentals*. Routledge, 2016.

[3] Frederick A.A. Kingdom and Nicolaas Prins. "Chapter 1 - Introduction and Aims." In: *Psychophysics (Second Edition)*. Ed. by Frederick A.A. Kingdom and Nicolaas Prins. Second Edition. San Diego: Academic Press, 2016, pp. 1–9. ISBN: 978-0-12-407156-8. DOI: https://doi.org/10.1016/B978-0-12-407156-8.00001-3. URL: https://www.sciencedirect.com/science/article/pii/B9780124071568000013.

[4] Frederick A.A. Kingdom and Nicolaas Prins. "Chapter 2 - Classifying Psychophysical Experiments." In: *Psychophysics (Second Edition)*. Ed. by Frederick A.A. Kingdom and Nicolaas Prins. Second Edition. San Diego: Academic Press, 2016, pp. 11–35. ISBN: 978-0-12-407156-8. DOI: https://doi.org/10.1016/B978-0-12-407156-8.00002-5. URL: https://www.sciencedirect.com/science/article/pii/B9780124071568000025.

[5] Shinichi Koyama. "[Application of psychophysics to neurology]." In: *Brain and nerve = Shinkei kenkyu no shinpo* 60.4 (Apr. 2008), 463—469. ISSN: 1881-6096. URL: http://europepmc.org/abstract/MED/18421988.

[6] Fabien Mathy and Mustapha Chekaf. "Experimenting with Psychtoolbox (and Others)." In: Jan. 2018, pp. 157–195. ISBN: 9781785482847. DOI: 10.1016/B978-1-78548-284-7.50008-3.

[7] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. "Towards Foveated Rendering for Gaze-Tracked Virtual Reality." In: *ACM Trans. Graph.* 35.6 (Dec. 2016). ISSN: 0730-0301. DOI: 10.1145/2980179.2980246. URL: https://doi.org/10.1145/2980179.2980246.

[8] Jonathan Peirce, Jeremy Gray, Sol Simpson, Michael MacAskill, Richard Höchenberger, Hiroyuki Sogo, Erik Kastman, and Jonas Lindeløv. "PsychoPy2: Experiments in behavior made easy." In: *Behavior Research Methods* 51 (Feb. 2019). DOI: 10.3758/s13428-018-01193-y.

[9] *PsychToolbox Official Website*. URL: http://psychtoolbox.org.

[10] Rolf Ulrich and Jeff Miller. "Threshold estimation in two-alternative forced-choice (2AFC) tasks: The Spearman-Kärber method." In: *Perception & psychophysics* 66 (May 2004), pp. 517–33. DOI: 10.3758/BF03194898.

[11] *Unity XR Manual*. URL: https://docs.unity3d.com/Manual/XR.html.

[12] Hong Ren Wu, Amy R. Reibman, Weisi Lin, Fernando Pereira, and Sheila S. Hemami. "Perceptual Visual Signal Compression and Transmission." In: *Proceedings of the IEEE* 101.9 (2013), pp. 2025–2043. DOI: 10.1109/JPROC.2013.2262911.