

---

# User guide for the *Ocular Dominance Experiment 2.21*

By Malcolm Handley, Michael Bevin, KeeKim Heng, and Robert O'Shea.

## 1 Introduction

### 1.1 The philosophy and the approach

#### 1.1.1 What ODE does

The Ocular Dominance Experiment program (ODE) displays stimuli on two monitor screens and records the user's responses via key presses. We designed ODE to run experiments in binocular rivalry; the two screens would normally be viewed through a stereoscope. Experiments comprise of one or more *trials*.

We designed ODE to work closely with StatView and with BBEdit Lite. StatView is a statistics program that can be used to analyse results from ODE (see 1.5). BBEdit Lite is a text (ASCII) editor. All input files for, and output files from, ODE are text files. All of these files have the same first name, differing in their extensions.

#### 1.1.2 How ODE does it

We specify all the experimental conditions that stay the same over an experiment in a *parameter* file (e.g., *expname.param*). We specify all the experimental conditions that vary within an experiment in a *trials* file (e.g., *expname.trials*).

ODE saves results into a *results* file (e.g., *expname.results*) that can be imported effortlessly into StatView for analysis. ODE generates a *counter* file (e.g., *expname.counter*) in which it keeps track of which trials the observer has completed. This means the observer can take a break in the middle of a block of trials, then come back and complete them, even if the computer has been shut down in the meantime. ODE can run two main types of trials: *rivalry* or *pseudorivalry* trials and *probe* trials.

#### 1.1.3 Rivalry and pseudorivalry trials

In an example rivalry trial, ODE could display for one minute a red vertical grating and a blue horizontal grating on one screen and a green disk on the other. The user could press three response keys to indicate which stimuli were visible. Another form of rivalry trial is a *pseudorivalry* trial, during which, for example, one stimulus (say a vertical grating) appears for a time on both screens, and is then replaced by another (say a horizontal grating), which is in turn replaced by the first, and so on, according to a prearranged sequence (contained in a *contrast-control* file). The observer in this case could press two keys to report visibility of the

---

stimuli. Rivalry and pseudorivalry trials can be intermixed in the one experiment. In both cases, ODE stores summary statistics about the user's key presses (total duration, total number of presses, and average press duration). As well, the program can create files describing exactly which keys the user was pressing at any time during a trial (key-press record files); these files can then be used to make contrast-control files for pseudorivalry sequences.

#### *1.1.4 Probe trials*

In an example probe trial, ODE can be made to show rival stimuli, and then introduce one or more instances of a probe stimulus during a trial to which the observer responds with a key-press. In its simplest form, the probe experiment could wait until the observer presses a single key to signal that he or she is ready, then, after a random time, show a spot to which the observer releases the key. ODE records the time between each onset of the probe and the release of the key. Probe and rivalry trials cannot be mixed in the one experiment. `{true?}`

#### *1.1.5 General experimental methods: Rivalry trials*

For rivalry trials, we normally have a trial duration of 60 seconds, and an inter-trial interval (ITI) of at least 45 seconds (meaning that, once time spent getting ready for the next trial is added in, the elapsed time is more like 60 seconds). In some of our experiments with complicated designs, one block might contain more than 50 trials, so to do 50 trials in one session would take  $2 \times 50 = 100$  minutes. We recommend, however, that observers limit each session to 60 minutes, and to have about 24 hours' rest between sessions. So two blocks of such an experiment ( $2 \times 100$  minutes) would take about four sessions to complete, one each for four days.

We normally construct our trials files so they contain one run through the design, with one trial per cell. This means we cannot statistically test our design completely until we have run at least two blocks of trials (i.e., `e1JW1.trials`, and `e1JW2.trials`). Usually we will do at least four replications of all crucial trials in an experiment. (Remember that each different trial file will require its own parameter file: i.e., `e1JW1.params`, and `e1JW2.params`.)

#### *1.1.6 General experimental methods: Probe trials*

For probe trials, the trial duration is determined by how long the observer takes to respond to all the probes. We normally present 25 probes in each trial. We use a hazard function for the random foreperiod, with the earliest probes appearing 500 ms after the observer's key press, and the latest probes appearing 7000 ms after the observer's key press. We normally use a key release for the observer's response. We normally set the minimum delay between a response and the ready signal for a new probe at 1000 ms.

---

## 1.2 Running ODE

### 1.2.1 Before starting

To run ODE, first set up the two screens with their desired resolutions, with the menu bar on the right screen. All ODE's required files must be in the same folder as ODE (see below).

### 1.2.2 Starting

Double-click on the ODE icon.

### 1.2.3 ODE's initial text window

A text window appears containing the words "Enter the name of your dataset:", at which you type the name of the experiment you want ODE to run (e.g., *expname*). Instructions on preparing these files are given below. Sample files for a probe experiment are given at the end of this document. If all that experiment's required files are available in the same folder (see below), ODE will check the trials.

### 1.2.4 ODE's error checking

ODE checks the parameter and trials files to make sure everything is ship-shape. There are three outcomes: If everything is fine, ODE displays *alignment screens* (see 1.2.5). If there are some classes of errors (e.g., luminance errors) ODE displays *warnings*, but it gives you a chance to continue. If there are other classes of errors, ODE identifies them, and refuses to run the trials (this is the usual outcome if you are trying to write your own parameter and trial files!). You can find more about error checking in section 8.

### 1.2.5 Alignment screens

Alignment screens are fields containing vertical nonius lines. This is to allow an observer to adjust the stereoscope to align the nonius lines. Once the observer is happy with the adjustments of the stereoscope, either clicking on the mouse button or pressing the return key makes ODE go onto the experiment's trials.

### 1.2.6 Pre-trial interval

ODE may take some time to generate the stimuli for a trial (the time depends on the size and complexity of the stimuli). Before this happens, a dark, *quit* screen appears for five seconds, with the words "**You may quit now**". Pressing command and the *quitkey* (see below) during this time exits the program. (The sample parameter file sets this to Q for a standard Macintosh keyboard.) Otherwise, this screen is followed by another dark, *preparation* screen

with the words “**Preparing to start trial 1**” or text of your choice (see below). Once the stimuli are prepared, the computer will beep, and make the nonrival parts of the display appear (i.e., fixation stimuli, fusion stimuli, backgrounds).

### 1.2.7 Rivalry trials

During rivalry trials, pressing then releasing all of the response keys that are defined for that trial starts the trial, making the gratings and pictures appear. The observer then would use the response keys to record his or her perception of the gratings and pictures. When the trial duration elapses, the rival stimuli are replaced by the inter-trial-interval screens.

While a trial is running, you may end it early by pressing command-*endtrialkey* or extend it by pressing *extendtrialkey* (it will end when *extendtrialkey* is released). (The sample parameter file sets these to escape and caps-lock, respectively, for a standard Macintosh keyboard.) Using either of these keys will cause -1s to be written out as the average, count and time for both the left and right keys.

If you pressed the *extendtrialkey* or the *endtrialkey*, ODE resumes with the next trial. To rerun the trial you missed, you need to quit the program, decrease by 1 the number in the `expname.counter` file (using any text editor such as BBEdit), then rerun ODE.

### 1.2.8 Probe trials

During probe trials, the rival gratings and pictures appear as soon as they are generated. Pressing any one of the keys that are defined for that trial presents the probe stimuli. The observer would then usually use the same key to give a response. Immediately after the observer's response to the probe, ODE can give one or more beeps. If you have asked for more than one presentation of the probe, it gives another beep or beeps when it is ready for another key press. The program can provide feedback to the observer through using a different number of beeps to represent a mistake or a correct response. When all probes for a particular trial have been responded to, the rival stimuli are replaced by the inter-trial-interval screens. To end any trial early, the observer needs to press command and the *endtrial* key.

### 1.2.9 Inter-trial interval

Once a trial finishes, either because the trial duration expires, or because all of the probe presentations have been responded to, the quit and preparation screens reappear (showing the upcoming trial number), and the observer can either quit, or continue until all the trials are completed.

### 1.2.10 Finishing

Once all the trials for an experiment have been completed, a dark screen appears with the words “**You have finished all  $n$  trials**”. Otherwise, you can press the quit key during the inter-trial interval.

## 2 Vital tips for running ODE

- Ensure that the menu bar is on the right-hand screen (scr=1).
- Quit from BBEdit before running ODE and quit from ODE before running BBEdit (otherwise a G3 Macintosh will crash).
- ~~Give more than one presentation of probes (see Known Bugs).~~

## 3 Known bugs

- ~~When running a multiple presentation probe experiment with a foreperiod determined by a hazard function, ODE can say mysteriously after probe presentation three that a crucial parameter is not specified (e.g., see test2.param and test2.trials). This exits the program without saving any data for that trial.~~
- Occasionally, after an especially long RT, ODE crashes the Macintosh.
- During probe trials, pressing the quitkey exits ODE without saving any data for that trial. (This is not really a bug; just a feature in which ODE’s ergonomics are not foolproof.)
- If you omit the reference for a restriction (e.g., you write “cent(fix) = 0” instead of “cent(fix=0) = 0”), ODE will sit forever after you enter the experiment name in its initial dialog box.
- Two keys must be defined for probe trials, even if only one of them is used to introduce and respond to the probe. (This is not really a bug; just a feature in which ODE’s ergonomics are not foolproof.)
- ODE leaves the number of colours of the monitors at whatever it used in the experiment, rather than returning them to their original state, to 256 colours (after it has finished running?).
- The *paste* option of *probecomb* does not seem perfect—some pixels do not have their correct luminances.

- 
- There are several bugs with using motiontf: A bug in the error checking means you cannot run trials in which gratings move differently on the same screen. You can bypass this bug by setting up a dummy first trial, then setting the counter file to contain 1. Another is that the starting phase of gratings seems to be required to be zero. Anything else means ODE sits forever when preparing the trial. Motiontf does not seem to work for probes. When everything on the screen is gray, motiontf uses 8-bit graphics. When colour is present, it uses 24-bit graphics.
  - Whenever any nonzero saturation is given for a particular screen, coloured tints (usually green) are produced on stimuli that should be gray. These differ on the two screens, presumably because the calibration files differ.
  - If the top part of any stimulus overflows the dimensions of the screen, none of it is drawn.
  - Screenshots cannot be taken when doing pseudorivalry.
  - The checking done before the experiment starts does not take account of the location of gratings and pictures in determining the validity of luminances. This means that two gratings or pictures will be treated as lying above each other, causing ODE to think that the luminances are much higher than they really are. You will need to use a low lumfac or trick the program to get around this.\
  - Occasionally, in experiments in which two response keys are used, StatView opens the results file with all the data on a single line of the file instead of on different lines. This can be fixed by adding StatView's missing data character (•) to the last two tabs of each line of the results file.
  - The ODE has problems with using the hazard function that can cause it to crash.
  - The final record in the key press record has been added twice, though it appears only once in the key press file. This works fine but the code is rather confusing. {Check that the state of the keys is recorded in this final record.}
  - In probe trials, the observer can sometimes use incorrect keys. See pfr1.param and pfr1.trials. In the first trial the observer is asked to press enter, but he or she can also press space.

---

## 4 ODE's files

### 4.1 Input files for ODE

All aspects of the experiment (e.g., what is displayed on the screens, for how long, their contrasts, and sizes) are controlled by the *values of symbols* that are found in the parameter and trials files. As well, there needs to be a *calibration* file for each screen. If pseudorivalry trials are included, their *contrast-control* files need to be present. If some of the stimuli are pictures, these files need to be present also. All needed files must be in the same folder as ODE. {Might be nice to have just one copy of ODE.}

#### 4.1.1 Parameter and trial files

Parameter and trial files must be named `expname.param` and `expname.trials` where `expname` is the name entered at the prompt. Make sure there are no stray characters in the file names (such as spaces). ODE does not, however, care about case.

#### 4.1.2 Calibration files

There also must be two calibration files, one called `calib table left (lm)` for the left monitor, and the other called `calib table right (lm)` for the right monitor. These calibration files should be those produced by the Smith Kettlewell Light Mouse (i.e., `lm`).

#### 4.1.3 Contrast-control files for pseudorivalry trials

Pseudorivalry trials are like movies, in which stimuli change physically on the screens in a rivalry-like way. The scripts for these movies are contained in contrast-control files. These can be from key-press-record files (see below).

#### 4.1.4 Picture files

Picture files must be of type PICT, square, with an even number of pixels, and with bit values ranging from 0 to 254.

### 4.2 Output files from ODE

ODE modifies, or produces, two text files for each trial: `expname.results` and `expname.counter`. If specified in the parameter or trials file, ODE will produce `expname/transpref/date time trialno` containing a record of all the key presses during a trial. These files will be in the same folder as ODE.

## 5 ODE and helper applications

### 5.1 StatView

#### 5.1.1 Using StatView to construct trials files

We normally generate a standard expname file in StatView (e.g., expname.sv). We use this file to produce our trials files. StatView is like a spreadsheet, with one line for each trial. Each factor in the design of the experiment is given one column, and the levels of the factor appear in that column. For example, if we had a two-factor design, in which we manipulated contrast over five levels (e.g., 0.1, 0.3, 0.5, 0.7, and 0.9), and the position of a fixation stimulus over two levels (i.e., 2 degrees to the left [-2] or 2 degrees too the right [2]), our expname.sv would contain 10 lines, representing the full combination of these two factors:

contrast	hcent(fix=1)
0.1	-2
0.3	-2
0.5	-2
0.7	-2
0.9	-2
0.1	2
0.3	2
0.5	2
0.7	2
0.9	2

#### 5.1.2 Using StatView to pretest an experimental design

With complicated designs (we have them up to four factors), we can use StatView to generate a column of random numbers as our measured variable, and do a test ANOVA to make sure the design is balanced.

#### 5.1.3 Using StatView to generate a random order

Once we are happy with our design, we can use the column of random numbers to sort the file, delete that column, then make a text version of the file called expname.trials (e.g., e1JW1.trials). Using StatView, we generate, sort, and delete a new column of random numbers each time we need to produce a new block of trials in a new trials file (i.e., e1JW2.trials). This ensures that the order of trials within each block is random.

#### 5.1.4 Using StatView to analyse results from ODE

Once we have our parameter and trials files set up, ODE can read them, and run the trials in their order in the file (i.e., in random order). It then writes into a results file all the information in the `exname.trials` file, one trial per line, with the data at the end of each line. This makes it simple to import the data back into StatView to analyse them.

#### 5.1.5 Using StatView file `Probe ex max` to calculate probabilities of probe catch trials

`Probe ex max` is a StatView file containing a formula that works out the probability of catch trials during probe experiments.

## 5.2 BBEdit [Lite]

We use BBEdit [Lite] to edit any of ODE's input or output files. ODE saves all text files with BBEdit as their creator, so double-clicking on any such file will open BBEdit. If you use the approach above of using StatView to create trials files, you can open them using the File menu of BBEdit.

## 5.3 Key Display 1.0

ODE refers to keys by keyboard codes. Key Display 1.0 is an application to show these codes.

## 5.4 Screen Info 1.0

ODE bases its calculations of visual angles on the number of pixels per meter of the screen. Screen Info 1.0 is an application to help you compute these values.

## 5.5 Stereograms

Stereograms is an application that draws random-dot stereograms on the monitor screens. This is helpful to check that an observer has normal binocular vision and is well converged in the stereoscope. **Note! Stereograms requires the menu bar to be on the left screen.**

# 6 Display issues

## 6.1 How ODE decides what to display

The way ODE decides what to show on a particular trial is complicated (see later), but essentially, information in the trials file takes precedence over information in the parameter file and later information takes precedence over earlier information. This is to avoid the necessity of

---

specifying everything about every particular symbol. For example, if a diameter is specified in the parameter file, it applies to every grating and picture that will be shown in the experiment, unless a particular grating or picture is given its own diameter later in the parameter file, or in the trials file.

## 6.2 Display order

ODE has a list of priorities for displaying things on screens. Fixation stimuli are pasted on top of everything else, with stimuli having lower ID values (e.g.,  $fix=0$ ) being pasted on top of stimuli having higher ID values (i.e.,  $fix=1$ ). Next, fusion stimuli are pasted, again with the higher priority assigned to lower ID values. Next, gratings and pictures are summed in various ways (see *combine*, *probecomb*, *lumfac*). When defining more than a couple of gratings and pictures, it is easy to exceed the maximum luminance a screen can produce.

## 6.3 Display luminances

Because each grating and picture comes with its own background, you need to be careful to ensure that the luminance and contrast you specify for a particular grating or picture are actually those you display. And all luminances will be reduced by reflection from the mirrors of the stereoscope; this is typically about 10% per mirror.

## 6.4 Probe timing

When presenting probes, the program redraws both screens. This always takes the same time, irrespective of the complexity of the stimuli, although more complex stimuli occupy more of the pretrial time. It does mean that a probe presented on the left screen appears before a probe presented on the right screen by whatever time is taken to draw one screen (about 5 ms?), giving left-screen probes a head start by this duration.

The probe remains on the screen for a defined time or until the observer makes a response (e.g., by releasing the key). Reaction times are recorded in milliseconds from when the last pixel of the probe is drawn until the observer makes a response.

Many probe presentations can be made in each trial. Probe presentations can be separated by a minimum duration called the inter-repeat interval (iri). This period of time commences after a probe presentation is over, and will not commence until all buttons have been released.

When the response is a CR, the probe is never presented to the observer. In a catch trial, the probe stimulus is also never presented to the observer.

---

## 7 The ins and outs of input and output files: A tutorial

### 7.1 The parameter file

The parameter file should contain all the values you want to be the same for all trials, such as always having a dark background (e.g., of 0.5 Cd/m):

For example:

```
back = 0.5
```

Each line should have a symbol (e.g., `diam`) followed by: an optional restriction in parentheses (see 9.2), optional tabs and spaces, an equals sign, optional tabs and spaces, and the value for that symbol. Blank lines are allowed. Lines that start with a hash (`#`) are ignored and can be used to insert comments into the file.

For example:

```
diam = 2  
diam= 2  
diam=2
```

Each line above gives every grating and picture a diameter of 2 degrees. As a convention, use one space around equals signs for values, but no spaces for restrictions (i.e., within parentheses). The example above in boldface follows this convention.

Symbols can have restrictions placed on them that will confine the symbol's influence to such things as a particular screen or stimulus.

For example:

```
diam(scr=0) = 2  
diam(scr=1) = 4
```

These two lines place restrictions on the diameter such that it will be 2 deg for the left screen, and 4 deg for the right screen. (Note in these examples our convention of eliminating all spaces within parentheses.)

### 7.2 The trials file

The first line of the trials file must contain the symbols (optionally with restrictions) that will

change for each trial and any parameters not already defined in the parameter file. Each symbol must be separated by at least one white-space character (a space or a tab). Restrictions on symbol name (appearing within parentheses) can contain spaces, but symbol names cannot. Again, by convention, we use no spaces within symbol names or their restrictions.) No quotation marks may appear in this line of the trials file (StatView 5 can insert quotation marks around symbol names).

Each subsequent line describes one trial as one row of a table.

For example:

SF(scr=0)	sf(scr=1)	Contrast	grating	meanL
5	1	.8	sine	50
5	1	.8	missing	30

On the first trial, a 5-cycle-per-degree, sine-wave grating (whose characteristics such as orientation, phase, and diameter had already been specified in the parameter file) would appear on the left screen (scr=0), and a 1-cycle-per-degree, sine-wave grating would appear on the right screen (scr=1). Both gratings would have a luminance of 50 Cd/sq-m. On trial 2, these sine-wave gratings would be replaced by missing-fundamental gratings having mean luminances of 30 Cd/sq m.

If you have any text containing spaces in any column of your trials file, such as pretext, you must place it double quotation marks, otherwise its spaces will be taken as field separators. To avoid using quotation marks, you can use `_the_underscore_character_instead_of_spaces`.

### 7.3 The results file

The program checks whether the results file exists. If not, it creates it and writes out the column headings. If it is present, the program appends the data to whatever results are already in the file. This happens at the end of each trial to ensure that the most data you lose if the computer crashes during a trial are just those for that trial. If you reset the counter file (see below), the trial numbers that are written into the results file will reset too, but these trials' data will still append. Appending data means that if you change the variables in the trials file without removing the results file, then the new data will not agree with the column headings at the top of the file.

If the `endtrialkey` or the `extendtrialkey` is pressed at any time during a trial, all the data are recorded as `-1s`.

#### 7.3.1 Rivalry/pseudorivalry trials

The `expname.results` file is the same as the trials file, but with added columns on the right: the

total time in milliseconds for which each of the four response keys were depressed, the number of presses for each response key, and the average time of each press for each response key. The average for each key is computed from the total time of all completed key press for that key. That is, if a key was pressed when the end of the trial occurred, the press would be added to the total presses for that key, the time would be added to the total time for that key, but this press and its time would not contribute to its average.

We always tell our observers to press no more than one key at a time. This is enforced by ODE in its key timing and counting. If more than one key is pressed simultaneously, whichever key was pressed first is counted. The second key is counted only after the first key is released. The first key has its offset recorded at the moment the other key is pressed. The second key has its onset recorded at the moment the first key is released and vice versa. For example, if key1 was pressed continuously for five seconds, and key2 was pressed twice for one second each, one second after key1 was pressed and with one second between key2's first offset and second onset, key1 would be recorded as being pressed three times, for one second each. Key2 would be recorded as never being pressed. If you require something other than this (e.g., durations of simultaneous presses), use key-press records (see below).

### 7.3.2 Probe trials

In the case of probe trials, the `expname.results` file is the same as the trials file, but with these added columns on the right: the delay between the first key press and the appearance of the probe, the reaction time (both in milliseconds), the response accuracy (whether the response was a Hit, Miss, False Alarm [FA], or Correct Rejection [CR]) ~~{Miss does not work}~~, and the number of the key that was pressed (e.g., if key1 were pressed, the program records a 1). Each presentation of the probe is recorded on a new line of the file; the trial information is repeated on every line.

In the case of a False Alarm, in a non-catch trial, a negative reaction time is presented. This is the time the observer reacted minus the time the probe would have taken to appear. In the case of a Miss, the reaction time is the probe miss time. In the case of catch trial, the delay time will be -1. This can be used as a flag to signal a catch trial in the results file.

## 7.4 The counter file

The `expname.counter` keeps track of how many trials the observer has completed. It contains a single number that is updated when the trial finishes. The program creates the counter file if it is missing. When the value in the counter file is zero, the program checks all the trials specified by the parameter and trials files. If it finds any errors (e.g., missing values, or values that are out of range), it prints these in the text window, and the program will not run until the trials and parameters pass its test (see Error Reporting, below). You can, however, bypass this testing

---

phase by setting the value in the counter file to any number greater than zero and less than the number of trials specified in the trials file. This is quite a dangerous thing to do. The program may crash if some illegal values are encountered. If any pixel's luminance is lower or higher than the screen can deliver, then that pixel will be given a random luminance, hue, and saturation.

## 7.5 Key-press record files

### 7.5.1 Rivalry/pseudorivalry trials

If the `transpref` symbol is defined in the parameter or trials file (e.g., “transcript”), for each trial ODE will write a file called `expname/transcr/date time trialno` containing a record of all the key presses during that trial. The program gets the short date and the time according to the format specified in the Date & Time control panel. How much of the `expname/transpref` you get depends on how many characters are spare after writing the date, time, and trial number into a file name that can contain, at most, 31 characters. The program will delete the rightmost characters of `expname/transpref`. The key-press-record file consists of at least three lines: the symbols from the results file, the data for that trial file from the results file, and the headings for the key-press record. Under that will be lines of five columns: the time in milliseconds that a response key changed state and the state of each of the four response keys (1 = pressed; 0 = not pressed).

## 8 ODE's error checking

When ODE loads, if the counter file contains a zero, it checks the parameter and trial files to make sure all needed symbols are present (e.g., that all gratings have a wave specified). If the trials pass that test, it goes on to check whether the values for each symbol are OK.

If any errors are found, they are reported and ODE will refuse to run the trials. One line is displayed for each error. Each line contains a specifier (in the same form as the restrictions entered in the experiment files) which shows what part of the experiment caused the problem. As well, ODE prints the maximum and minimum luminances (for all guns combined, and each separately) able to be produced by the offending screen. This is useful only when the error was because a luminance was asked for that is outside this range {print out these only if useful???}. Luminance errors are actually *warnings*: ODE will run, but with the luminances of erroneous pixels chosen randomly. This could happen because of `meanL`, `contrast`, `lumfac`, or `combine`. Changing these values in the parameter or trials file should fix the problem. For complicated stimuli that may combine several gratings, or for pseudorivalry trials that combine stimuli normally shown on separate screens, how luminances sum is governed by the symbol `lumfac`. You will normally get what you expect if you set `lumfac` equal to  $1/(\text{number of stimuli to be combined})$ . Luminances that are out of range generate warnings so that you can ignore them if you are confident that they are spurious.

## 9 ODE's values and symbols

### 9.1 What ODE can do with values and symbols

This is to give you an overview of what ODE can do and to tell you where to look to find out how to do it. ODE can:

- Produce and display many different sorts of gratings and pictures on each screen. Each grating and picture has its own background, mean luminance, and contrast See the *Chroma*, *Positioning*, *Gratings*, and *Pictures* sections of *The Symbols*.
- Produce and display fusion stimuli. See *Positioning*, *Chroma*, and *Fusion stimuli* in *The Symbols*.
- Produce and display fixation stimuli. See *Chroma* and *Fixation stimuli* in *The Symbols*.
- Produce and display pseudorivalry. Either the contents of both screens will be displayed on each screen, or each grating and picture will be displayed only on its own screen, with the contrast of the other reduced to zero. The contrast of the gratings and pictures will be adjusted according to a contrast-control file (see *Pseudorivalry* in *The Symbols*).
- Produce and deliver probe gratings and pictures during rivalry, and measure responses and response times.
- Collect key-press records of the observer's responses to each trial. See *General* in *The Symbols*.
- Produce screenshots. See *Key control* in *The Symbols*.
- Produce different sounds for different events. See *Sound* in *The Symbols*.

### 9.2 Restrictions on symbols

Symbols can be restricted to:

scr	screen. There are always two screens, numbered 0 for the left and 1 for the right.
grat	grating
pict	picture
fus	fusion stimulus
fix	fixation stimulus

Each of these restrictions must have an identifying number, which we will refer to as its ID. IDs for all restrictions other than screens can be any integer greater than or equal to 0 (e.g., stim=1). Screens can have IDs of 0 (left screen) and 1 (right screen) only.

### 9.3 The symbols

The following is a list of all the valid symbols and the values they can take. Symbols and their values are case insensitive.

All luminances are in Cd/m<sup>2</sup> and are real numbers.

All sizes are in degrees of visual angle and are real numbers.

#### 9.3.1 General

`odeversion`

optional, if set specifies the version of ODE (as a real with two decimal places; e.g., 2.21) for which the experiment was designed. ODE will then check this value against the running version and output a warning if the two are different.

`trialKind`

string. A trial may be one of four kinds.

- `normal`: Each screen's stimuli are displayed on the correct screen.
- `1imagepr`: You get a pseudorivalry sequence determined by the values in a contrast-control file. In this case, the same image is displayed on both screens (hence the name "one image"). This has the restrictions that the image must be within the luminance range of both screens and must be grayscale.
- `2imagepr`: You get a pseudorivalry sequence determined by the values in a contrast-control file in the same way as for `1imagepr`. However, stimuli are drawn only on their correct screens. This means that the observer will experience some real rivalry as well as the pseudorivalry. Because of this you should minimize the time during which both rival stimuli are visible. The benefits of doing this type of rivalry are that it allows you use colour stimuli in pseudorivalry and that the images are not limited to the intersection of the screens' luminance ranges.
- `probe`: Rival stimuli are displayed continuously. At some time after the observer presses a key, a probe stimulus is shown, and the observer's reaction time to detect the probe (signalled by a releasing the key, or by a new key press) is recorded.

#### 9.3.2 Key control

The keys that the observer uses to control the program can be customized. These are specified as the virtual key code of the key. You can find it by using the Key Display program. Different key symbols can be given the same key code, but this causes all the data to be written out as zeros (fix!). Extending a trial or ending it early will set its statistical information to -1 for each field.

key1  
key2  
key3  
key4

the response keys for responding to the stimuli.

Traditionally we have used the top left key on the keyboard (') for reporting one rival stimulus and the top right key on the keyboard (delete) for reporting the other. To get these keys on an ADB keyboard, use

key1=50  
key2=51

quitkey  
endtrialkey  
screenshotkey

the keys to press, along with command, to quit the program, to end a trial early, or to take a screen shot, respectively. These three attributes need not be specified. They default to being unusable. Useful values are Q (virtual key code = 12), escape (virtual key code = 53), and C (virtual key code = 8), respectively. These key codes are for extended ADB keyboards and may be different on USB (iMac and Blue and White G3) keyboards.

extendtrialkey

while this key is held down, the trial will not end. This is useful for looking at an image for as long as desired. When the key is released, the trial will end regardless of whether the trial duration has elapsed. Caps lock (virtual key code of 57) is a good choice for the extendtrialkey with ADB keyboards. With USB keyboards, we use the *h* (for “hold”) key.

### 9.3.3 Alignment screens

alignLineLum  
alignBackLum

real, the luminance of the lines and background in the alignment screen.

### 9.3.3 Trial

iti and trialLength

iti is the inter-trial interval. This is the number of seconds *before* the trial, so you could call it the pre-trial interval. trialLength is the length of the trial in seconds; it is ignored for probe trials. iti is ignored for the first trial.

**preText**

string. Put it in double quotation marks if there are any spaces in the string. This controls the text displayed while the trial is being prepared. If it is specified then the trial number will be prepended, otherwise the text will be “Preparing to start trial  $n$ ”.

**9.3.4 Screens****viewdist**

real, the distance in meters that the observer will view the stimuli from. This may change from trial to trial; the observer can be told to move to the new distance by using pretext.

**scrres**

the resolution of the screen in pixels per meter. This can be obtained by measuring the horizontal extent of the visible area of the screen in meters ( $x$ ), then dividing the number of pixels across the screen by  $x$ . For example, if  $x = .31398$ , and the screen shows 1152 pixels horizontally, then the scrres = 3669. With a viewing distance of 1 m, the pixel at the center of the screen with this scrres subtends approximately 0.0156 degrees, or 64 pixels per degree. This is a useful number because it ensures that you can specify spatial frequencies that are powers of 2 (e.g., 0.25, .5, 1, 2, 4 cycles per degree), allowing an integer number of pixels per cycle of any grating, the minimum requirement for a grating to be drawn without any aliasing. Most gratings have more stringent requirements, given in Section:sf. The vertical dimensions of the pixels should also be checked; the monitor should be adjusted until the screen resolution for the horizontal and vertical dimensions is the same. There is a program called Screen Info designed to help you with the drudgery of these calculations.

**combine**

gratings and pictures on a screen may be combined in one of several ways:

“sum” At each pixel, the luminances of all gratings and pictures (and their backgrounds) will be summed. This was the only method before ODE 2.9.

“min” At each pixel, the luminances of all gratings and pictures (and their backgrounds) will be compared, and whichever is the minimum will be displayed.

“max” At each pixel, the luminances of all gratings and pictures (and their backgrounds) will be compared, and whichever is the maximum will be displayed.

Note that in cases where a grating or a picture is positioned so that it falls on the background of another grating or picture, it will be combined with that background. This means that using max would prevent one stimulus from going dimmer than the background of the other stimulus. Using min has the opposite result.

### probecomb

when a probe experiment is running, the probe stimuli's luminances may be added to those of the stimuli on which they fall, or the probe stimuli may be pasted on top of the rival stimuli:

“normal” Combine just as if it was a normal stimulus using whatever setting was provided for combine. This is the default.

“paste” Ignore the non-probe stimuli, and paint the probe on top of whatever is displayed on the screen. {This paste operation does not seem perfect—some pixels do not have their correct luminances.}

### 9.3.5 *Key-press record*

A key-press record is a record of which keys were pressed during the trial. It contains enough information for the program to run a pseudorivalry trial that should produce a very similar response in the observer.

### transpref

the prefix of the name of the key-press record file generated for a trial. If this symbol is not specified then no file will be created. Note that file names on the Macintosh cannot contain colons (':'). They will be replaced with periods if you use them in a key-press record prefix.

### 9.3.6 *Chroma*

The hue and saturation of gratings, pictures, fixation stimuli and fusion bars can all be controlled with these symbols. They do not have default values but you can easily add two lines to the parameter file to make everything gray (hue = 0, sat = 0).

### hue

a real from 0 to 360 degrees indicating the desired hue on a colour wheel (see Figure 1).

### sat

the saturation of the colour. A real from 0 to 1, where 0 is gray and 1 is a pure colour. In Figure 1, the saturation is 0 at the center and 1 at the edge.

(If you use the system's colour picker to make your stimuli as PICTs, you also specify *value*. This gives a PICT a particular luminance. For the stimuli that ODE constructs, however, you specify luminance as explained in the *Stimuli* section.)



**Figure 1. Showing the Macintosh colour circle.**

### 9.3.7 Positioning

The positions of gratings, pictures, fixation stimuli, and fusion stimuli are controlled with these symbols.

hCent

vCent

the center of gratings, pictures, fixation stimuli, and fusion stimuli is controlled with these symbols. They specify the distance horizontally and vertically that the center of the item should be from the center of the screen, in cartesian coordinates (positive is right and up), in degrees. For example, vCent(fix=0)=2 will position a particular fixation stimulus 2 degrees to the right of the centre of the screen.

### 9.3.8 Gratings or pictures

There may be more than one grating or picture on each screen, each with differing characteristics. Gratings and pictures have their colours controlled with the symbols in the colour section and their positions controlled with the symbols in the positioning section.

The important difference between rival stimuli and other stimuli on the screen (fusion stimuli and fixation stimuli) is that each grating and picture has its own background, mean luminance, and contrast. Gratings and pictures also interact with each other (see combine in *Screen*), unlike fusion stimuli and fixation stimuli, which are simply painted onto the screen.

stimtype

for probe trials, specifies whether a particular grating or picture is a probe stimulus. A value of “probe” specifies a probe stimulus. Default is that a stimulus is not a probe.

---

For example, `stimtype(grat=1,scr=0)=probe` means that the probe is a grating with characteristics specified elsewhere for (`grat=1`) that will be presented on screen 0.

#### back

the intensity of the background of a grating or picture. This value may vary between stimuli on the same screen and controls that luminance towards which the stimulus is blurred.

#### blurshape

the shape of the blur. Any of:

“oval” for radial blurs.

“rect” for rectangular blurs.

The fields of gratings and pictures are still square and so you will actually get square and circular blurs. {This will change in the future?}

#### blur

the type of blur to apply across the radius of the field. Any of:

“none” for no blurring.

“ramp” for a straight ramp.

“gauss” for a non-cumulative gaussian blur.

“cpd” for a cumulative gaussian blur.

Note that you will almost certainly want a rectangular blur shape and no blurring for pictures.

#### blurProp

the proportion of the grating or picture to be blurred (e.g., 0.25 will blur the outer quarter of the field). Any real number from 0 to 1.

#### contrast

the Michelson contrast. It can be any real number from 0 to 1.

#### lumFac

Depending on *combine* and *lumfac*, when multiple gratings and pictures are on one screen they can be added in their entirety. Even their backgrounds are added. It is as though each stimulus has been projected onto a screen by its own slide projector, so the more stimuli you define, the more light will be projected onto the screen. It is easy, therefore, to exceed the maximum luminance that a screen can produce. To simulate the reduction in luminance which is caused when stimuli are added optically by half-silvered mirrors, all stimulus luminances are multiplied by *lumFac* before they are put on the screen. Thus you can simulate averaging, should you want, by setting *lumFac* to  $1/(\text{number of gratings} + \text{pictures})$ .

If you are doing psuedorivalry, the luminance factor for screen 0 will be used for all

gratings and pictures.

meanL

the mean luminance of a grating or picture. In the case of a picture, bit values of 128 are given the meanL, and 0 and 254 are given luminances determined by the contrast and the meanL. The luminances of other pixels are interpolated between these anchor points. Using bit values of 255 will produce unpredictable results and should be avoided. This range was chosen so that there would be an odd number of values, giving a clear mid value.

### 9.3.9 Gratings

**Note: in order to get a grating to appear, you need to mention an ID for it in either the param or trials file [e.g., wave(grat=0)=sine].**

The chroma of gratings is controlled with the normal chroma controls. See *Chroma*.

The position of gratings is controlled with the normal positioning controls. See *Positioning*.

angle

the angle of the grating in degrees measured clockwise from 0 as vertical. An angle of 0 means that the bars will be vertical.  
0 to 360. Integer.

diam

the diameter of the grating.

wave

the luminance profile of a grating:

“flat” for a disk of the mean luminance.

“sine” for a sine wave.

“square” for a square wave.

“tlsaw” for a saw tooth with the dark section at a phase value of zero.

“brsaw” for the opposite to “tlsaw”.

“missing” for a missing fundamental.

wavecoord

string, one of two values

“normal” - waves displayed normally

“polar” - the waves are displayed as concentric circles or radially, this effect being produced by converting the co-ordinate axes from cartesian to polar. A horizontal wave thus becomes concentric circles, and a vertical wave becomes spokes. The program automatically centres the grating assuming the offset {phase?} is zero. {The spatial

frequencies of the radial and concentric gratings differ as do their horizontal centres.  
The radial grating has a discontinuity along the horizontal meridian.  
 “spiral” {yet to be implemented}

modWave

modPhase

modSf

the wave in a grating (let’s call it the carrier) can be contrast-modulated by another wave (let’s call it the modulator). Currently, the carrier is modulated in contrast between 0 and its specified contrast. These symbols specify, respectively, the waveform, phase, and spatial frequency of the modulator. You must specify all three of these symbols. If you don’t specify the wave, ODE will not notice that you are trying to modulate and will not even give you error messages. In future versions of the program, the minimum modulation contrast will be able to be set to some value between zero and the contrast of the carrier.

phase

the phase of the grating in degrees with reference to the center of the field. It can take any integer between 0 and 360. At zero phase, assuming vertical gratings and reading from left to right we get:

“sine”	starts at mid luminance, then goes lighter
“square”	lighter for first 180 deg, then darker.
“tIsaw”	darker then lightening.
“brsaw”	lighter then darkening.
“missing”	lighter for first 180 deg, then darker.

Phases other than zero mean that each wave starts that many degrees into its cycle (e.g., 45 degrees phase would yield a sine wave with its maximum lightness in the center of the field).

sf

the spatial frequency of a grating, measured in cycles per degree of visual angle at the viewing distance. In general, SF may be any real number, preferably one that produces an integer number of pixels per wave. This means that it should divide the number of pixels per degree to give an integer. If this were 60 (we recommend 64!), a spatial frequency of 7 cycles per degree would not give an integer number of pixels, but an SF of 1.5 (giving 40 pixels per cycle) would. There is no requirement that waves have an integer number of pixels per cycle. You will be given a warning after the experiment has been checked, {true?} but may still run the experiment.

Some waves have more stringent restrictions on the number of pixels per cycle:

- sine waves and square waves prefer an even number of pixels.

- missing fundamentals prefer  $2k+4$  pixels, where  $k$  is an integer. This is so that there are four pixels for the four max and min points and an even number of pixels to be spread out between them.
  - flat waves and sawtooth waves have no special restriction.
- Failing to give waves their preferred number of pixels results in warnings, but the experiment can still be run.

#### motiontf

the phase of all gratings on a particular screen can change during a trial. motiontf is the number of cycles per second by which the phase should change. For gratings at zero degrees, positive values will cause the bars to move to the right. For gratings at 90 degrees, positive values will cause the bars to move down. **There is a bug in the error checking that means you cannot run trials in which gratings move differently on the same screen. You can bypass this bug by setting up a dummy first trial, then setting the counter file to contain 1. There is another bug; the phase of gratings seems to be required to be zero. Anything else means ODE sits forever when preparing the trial. Motiontf does not seem to work for probes.**

When everything on the screen is gray, using motion will mean that you get 8-bit graphics (as opposed to the usual 24-bit). When colour is present, you still get 24-bit graphics.

If there are more than one grating moving on the same screen then the number of frames required for an animation will be the lowest common multiple of the number of frames that each grating needs. The number that a grating needs is a function of the size of the phase shift per second and the frame rate. For example, if a grating moves 5 cycles per second and the program is generating 20 frames per second then the animation will repeat after 0.2 of a second so 4 frames will be needed.

Use a value of zero to indicate no motion.

#### fps

the number of frames per second to use for trials involving motion. This value is ignored (but still required) when there is no motion. The value of this symbol can vary over screens and trials [e.g.,  $\text{fps}(\text{scr}=1) = 30$ ] **but not over gratings**. The fps cannot usefully be greater than half of the refresh rate of the monitors. The higher the fps, the smoother the movement, but the greater the memory requirements and time to create a trial.

#### 9.3.10 Pictures

**Note: in order to get a picture to appear, you need to mention an ID for it in either the param or trials file [e.g.,  $\text{pictName}(\text{pict}=1)=\text{whatever}$ ].**

The position of the picture is controlled by the general position controls:hCent and vCent [e.g.,

$hCent(pict=1) = .5$ ; and  $vCent(pict=1) = 0$  would centre the picture half a degree to the right of the centre of the screen]. Pictures must be in PICT files in the same folder as ODE. PICTs must be grayscale, have bit values ranging anywhere within 0 and **254**, must be square, and have an even number of pixels per side.

The chroma of pictures is controlled with the normal chroma controls. See *Chroma*.

The position of pictures is controlled with the normal positioning controls. See *Positioning*.

Pictures are displayed so that one pixel in the picture equals one pixel on the screen.

**Bug-workaround 1: If you define any picture, it MUST appear on both screens. To make a picture appear on one screen only, give the one on the un-needed screen a meanLum of zero. This is true for every picture you define.**

**Warning: When using a graphics program to produce PICTs, it's tempting to use white (bit value = 255). But this value is out of range for ODE--use a bit value of 254 instead.**

pictName

the name of the picture file to be displayed. Enclose the name in double quotation marks if it contains spaces. Note that double quotation marks may not appear in the file name, either quoted or unquoted. To avoid displaying any picture, enter an empty name: "".  
{ **This does not appear to work.** }

### 9.3.11 Fusion stimuli

**Note: in order to get a fusion stimulus to appear, you need to mention an ID for it in either the param or trials file [e.g.,  $fuslum(fus=0)=100$ ].**

Fusion stimuli are unfilled ovals or rectangles of varying sizes, shapes, and luminances. They are centered on the screen, and are drawn on top of the gratings and pictures but behind fixation stimuli. Fusion stimuli with lower IDs appear on the top of those with higher IDs. Figure 2 show how fusion stimuli can be manipulated. For example, a rectangular stimulus can be made to appear as two vertical bars, as four bars all overlapping to resemble a hash sign; or even as a single, filled square.

The chroma of fusion stimuli is controlled with the normal chroma controls. See *Chroma*.

The position of fusion stimuli is controlled with the normal positioning controls. See *Positioning*.

fusKind

the shape of a fusion stimulus. One of:

“oval”

“rect”

“none”

Use none to make the stimulus invisible.

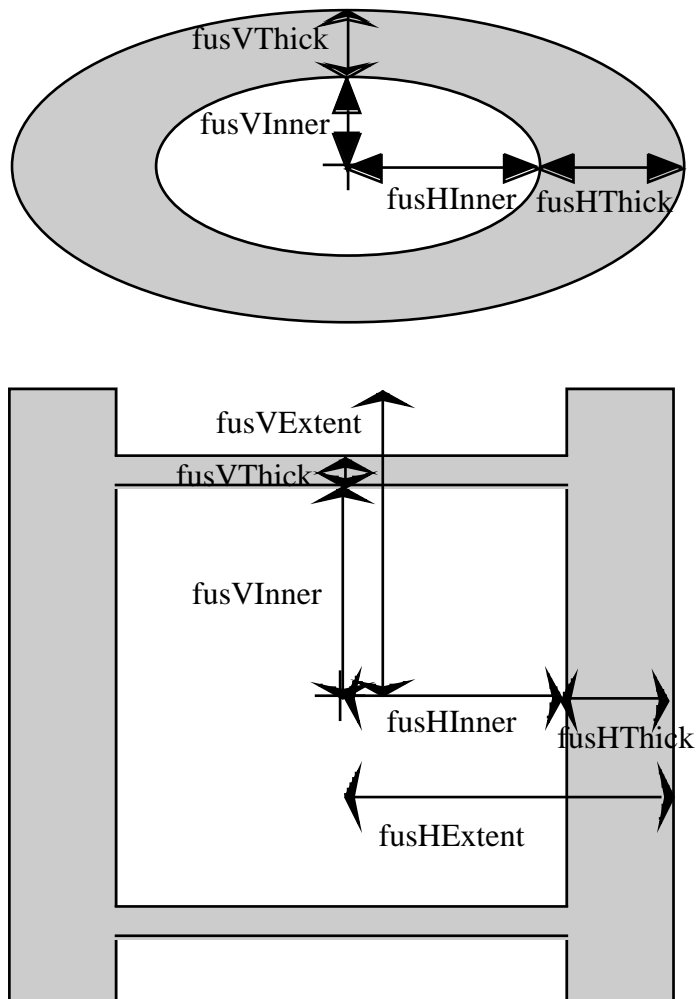
fusLum

the luminance of the fusion stimulus.

fusHInner

fusVInner

the horizontal and vertical distances of the inside edge of the fusion stimulus from its center.



**Figure 2. Showing the symbols for specifying fusion stimuli.**

fusHThick

fusVThick

the horizontal and vertical thickness of the fusion stimulus. If you are using rectangular fusion stimuli then you can remove the horizontal or vertical bars by using this symbol. For example, to specify two vertical bars, one on each side of a grating or picture, you may set the the vertical thickness of a rectangular fusion stimulus to zero.

fusHExtent

fusVExtent

fusion rectangles can be made into hashes by extending the horizontal and vertical bars. These symbols, which do nothing for ovals and are optional for rectangles, allow you to specify the distance between the centre of the stimulus and the ends of a bar. You can also use these to shorten the bars to produce disconnected bars or eliminate them all together. For example, to specify two vertical bars, one on each side of a grating or picture, you may set the horizontal extent of a rectangular fusion stimulus to zero. The extent is measured from the center of the screen.

### 9.3.12 Fixation stimuli

**Note: in order to get a fixation stimulus to appear, you need to mention an ID for it in either the param or trials file [e.g., `fixkind(fix=1)=cross`].**

The chroma of fixation stimuli is controlled with the normal chroma controls. See *Chroma*.

The position of fixation stimuli is controlled with the normal positioning controls. See *Positioning*.

fixAngle

the orientation of the fixation stimulus (for those where it matters: crosses and lines). Specified the same way as angle for gratings.

fixKind

the kind of fixation stimulus to draw. One of

“none”

“point”

“cross”

“line”

fixLum

the luminance of the fixation stimulus

fixRadius

the radius of the fixation stimulus, ie, the distance that it extends from its center.

fixThick

the thickness of the cross or line.

fixInner

the inner distance of the line from its center. This is ignored for non-lines.

### 9.3.13 Pseudorivalry

During pseudorivalry trials, rival stimuli that would normally be displayed simultaneously on separate screens are displayed one at a time on both screens, or on one screen at a time so that (ideally!) no rivalry is experienced. To simulate rivalry, stimuli change their contrasts according to a script contained in a contrast-control file. Key presses made in response to pseudorivalry are recorded in the usual way and can be saved in a key-press record file.

There are two types of pseudorivalry trial, obtained by setting `trialkind` to `1imagepr` (one-image PseudoRivalry) or `2imagepr` (two-image PseudoRivalry). In `1imagepr`, rivalry stimuli are combined and displayed on both screens. By setting the contrast on one stimulus to zero, the other is therefore visible to both eyes. By swapping the contrast, the visibility of the two stimuli can alternate to simulate rivalry. The combined image must be within the luminance range of both screens and must be grayscale. In `2imagepr`, each stimulus is drawn only on its own screen, and rivalry is defeated by setting the contrast of one stimulus to zero. This technique removes the restrictions of `1imagepr`, but adds the problem that any time neither stimulus is intended to be exclusively visible, both stimuli are visible dichoptically, creating the conditions for real binocular rivalry.

Contrast of rival stimuli are adjusted in tandem according to a contrast-control file. Where the file indicates exclusive visibility, the visible stimulus will be at full contrast (that is, its normal contrast) and the other be at zero contrast. During periods of mixed visibility, the contrasts will be ramped over the period specified by `prRampTime`, except where two ramps overlap (for example, half of the stimuli are ramping down, the other half up). When overlaps occur for the same stimuli (when they are meant to start ramping down before they have finished ramping up, for example), each ramp is compressed so that they no longer overlap. When they occur between the two sets of stimuli, the ramps are changed so that they overlap completely.

Caveats with pseudorivalry:

- Screenshots cannot be taken when doing pseudorivalry.
- Coloured pseudorivalry stimuli can be obtained only with the `2imagepr` technique.

`prContControl`

the name of the file to use to control the contrast of the two screens' images. This is only needed if you are doing pseudorivalry. The file is best constructed by editing a key-press record file; the first line is ignored, the second line must contain the headers for the columns separated by tabs (Time(ms), [key]1, 2, 3, and 4,), and the third line must contain a time, and 1s or 0s in the remaining columns, again separated by tabs. A "1" means that the particular grating or picture has its usual contrast, a "0" means that the particular grating or picture has zero contrast. During `1imagepr`, setting both rival stimuli to 1 will optically superimpose the stimuli. Setting both rival stimuli to zero will display a gray field {true?}. During `2imagepr`, setting both stimuli to 1 will create a

normal rivalry display, as will setting both stimuli to zero.

#### prNumFrames

the maximum number of frames to use in a transition from high to low contrast, or vice versa, including the start and finish frames (meaning that you must set this to at least 2). This controls memory usage and the time taken to prepare for a trial. A good number might be 10. This is only needed if you are doing pseudorivalry.

#### prRampTime

the time, in milliseconds, that a rival stimuli takes to change contrast completely. Ramps will take less than this time when there are other changes in visibility happening soon after or before them. A good number might be 500. This is only needed if you are doing pseudorivalry.

#### prIgnore

when this has a value of 1, this indicates that the program should not perform pseudorivalry on a particular grating or picture. When this symbol is zero or undefined (not present), then the stimulus will change in contrast over time, as per usual.

### 9.3.14 Probes

#### probefntype

~~can be set either to hazard or to random. These affect the delay between when an observer signals that he or she is ready (e.g., by a key press) and when the probe is presented. If hazard is used, the program implements the delay according to a hazard function, with probehazardmin specifying the minimum delay, and probehazardprob specifying the probability of the appearance of the probe. If random is used, the program implements a randomly delay with probedelayvariance around probedelay.~~

can be set to a hazard function or a random function. These functions are used to determine the delay between when an observer signals that he or she is ready (e.g., by a key press) and when the probe stimulus is presented to the observer. With a hazard function, the probability that a probe is about to be presented is always equal to the what you set (in probehazardprob), whereas with random, the probability that a probe is about to be presented grows with time elapsed in the foreperiod.

hazard: ODE determines the delay using a hazard function. Three additional symbols are required when using this function. These are probehazardmin, probehazardmax, and probehazardprob.

random: ODE determines the delay randomly. Three additional symbols are required when using this function. These are probedelayvariance, probedelay, and probeprob.

probehazardmin

integer; the minimum delay time in milliseconds when a hazard function is used

probehazardmax

integer; the maximum delay in milliseconds when a hazard function is used. If this time is exceeded, ~~rather than present the probe~~ the trial becomes a catch trial (this is how catch trials are generated by the hazard function), and the program simply moves to the next trial. This prevents the observer being forced to wait for a very long time on the occasions that the hazard function exceeds probehazardmax.

probehazardprob

real; probability at each millisecond that the probe is presented when a hazard function is used. This should be quite a low number. Use the following formula:

$$p(\text{Catch trial}) = (1 - \text{probehazardprob})(\text{probehazardmax} - \text{probehazardmin})$$

For example, for a trial with a maximum time of 7 seconds, a probehazardprob of 0.0004 yields a probability that the maximum delay will be exceeded of about 0.0685. Use the helper application *Probe ex max* to work out this formula.

probedelay

integer; the average delay in milliseconds when a random function is used.

probedelayvariance

integer; the range of probedelay in milliseconds when a random function is used. For example, if probedelay were 2000, and probedelayvariance were 500, then delays would range between 1500 and 2500 milliseconds. If the probedelayvariance were 0 then the delay would always be the value of probedelay. (For example, to make a random foreperiod that begins 300 ms after a key press is begun, and that lasts for 5000 ms, use  $300 + .5 * 5000$  and ensure that  $\text{probedelayvariance} = .5 * 5000$ .)

probeprob

real; the probability a probe is presented when using a random function is used. The value of probeprob must range between 0 and 1. This allows catch trials to be included when using a random function for the probe delay.

probemisstime

integer; specifies the time required to miss a probe presentation. This is required for both hazard and random functions.

repeats

integer; this parameter currently only applies to probe trials and is optional. If set for a

trial, the trial is repeated  $n$  times, without the normal inter-trial interval and screens in between. When the probe response is given, the probe disappears, and the iri happens. If the trial is finished too, then the rival stimuli disappear and the inter-trial interval happens.

#### probekeyhandling

string; there are two types:

release - the response key is pressed when the observer is ready, and released when he or she observes the probe.

secondpress - the response key is pressed and released when the observer is ready, and pressed and released again when the observer detects the probe. Times are taken from the pressing of the key rather than the release. The two key presses must be the same key (any other key presses for the second key press will be ignored).

#### iri

integer; this parameter specifies the length (in milliseconds) of the Inter-Repeat Interval, the minimum time between successive presentations of the probe. The default break is none. ~~If iri is specified, ODE gives a single beep after a response, and gives another beep after the iri has elapsed. If no iri is specified, ODE beeps twice after a response. The probe is always presented during the iri (I think), whatever the observer's response~~

#### probemode

string; there are two modes that can be chosen. the first mode is continuous. This causes the probe to appear, and remain on the screen until a hit or a miss. The second mode is flash. This causes the probe to flash on and off the screen. The duration of the probe presentation needs to be specified using probeflashtime.

#### probeflashtime

integer; this specifies the amount of time in milliseconds the probe stimulus appears if probemode has been set to flash mode.

### 9.3.15 *Sound*

Sound features apply only to probe trials, as a way of giving feedback to the observer. In probe trials, probesoundduration and probesoundfrequency must be defined, but the other sound commands are optional. To see the sound symbols in action, take a look at the sample files in **Appendix C**.

#### probesoundduration

integer; the duration in ticks (one tick is one sixtieth of a second) of a single beep emitted by ODE and of the pause between consecutive beeps. The value of 10 is

reasonable. A value of 1 does not produce any sound!

#### probesoundfrequency

integer; the frequency in cycles per tick (one tick is one sixtieth of a second) {?} of a single beep emitted by the ODE. The value of 60 is reasonable.

#### probearstartsound

string; there are three types

this is the sound made at the beginning of each trial repetition. The current options are onebeep, twobeep, threebeep or none. If this symbol is not defined, a default value of none {fix!} will be used.

#### probehitsound

string; there are three types

this is the sound made if the response is a hit. The current options are onebeep, twobeep, threebeep or none. If this symbol is not defined, a default value of none will be used.

#### probemisssound

string; there are three types

this is the sound made if the response is a miss. The current options are onebeep, twobeep, threebeep or none. If this symbol is not defined, a default value of none will be used.

#### probecrsound

string; there are three types

this is the sound made if the response is a Correct Rejection. The current options are onebeep, twobeep, threebeep or none. If this symbol is not defined, a default value of none will be used.

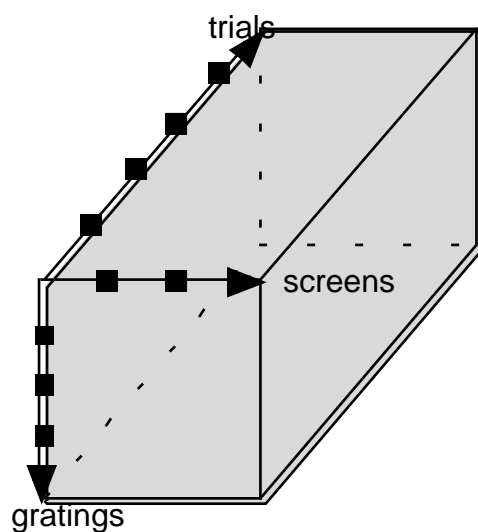
#### probefasound

string; there are three types

this is the sound made if the response is a False Alarm. The current options are onebeep, twobeep, threebeep or none. If this symbol is not defined, a default value of none will be used.

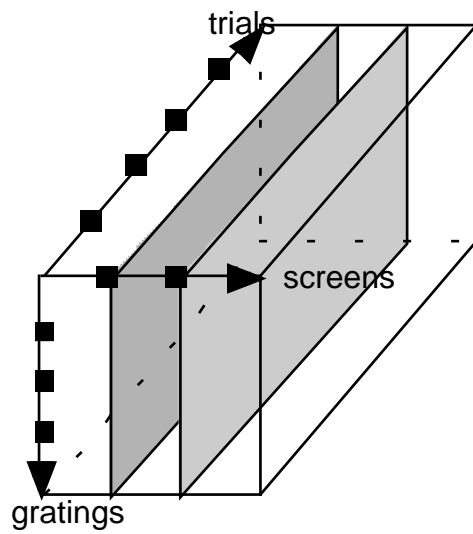
## Appendix A: Understanding restrictions

A useful way to think of restrictions is as axes in multidimensional space. Thus an experiment has a variety of axes. Think of an experiment that uses two screens, three gratings on each screen and four trials. You have three axes: screens, stimuli, and trials. Let's assume that you are varying the contrast of the gratings. You might start off wanting all the gratings in the experiment to have the same contrast. You might picture this like Figure 1, with the shaded part showing where the contrast symbol applies.

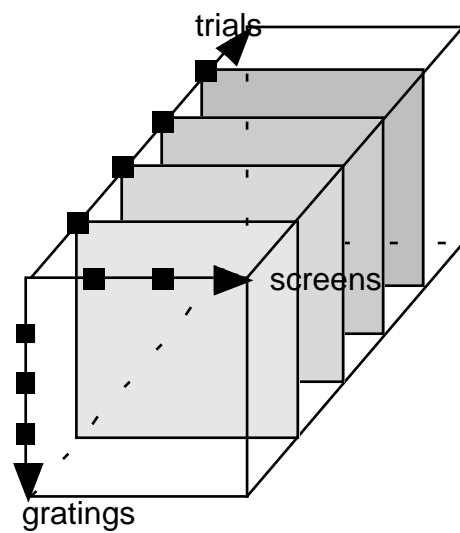


**Figure A-1.**

That is not much good, though. The observer will see the same thing on each screen. You could instead make two different contrast values, one applying to each screen. Now all of the gratings on the same screen will still have the same contrast as each other, regardless of the trial that they are in, but the two screens will be different (Figure 2). Alternatively, you could have a different contrast for each trial, but the same contrast on each screen during a trial. That would look like Figure 3.

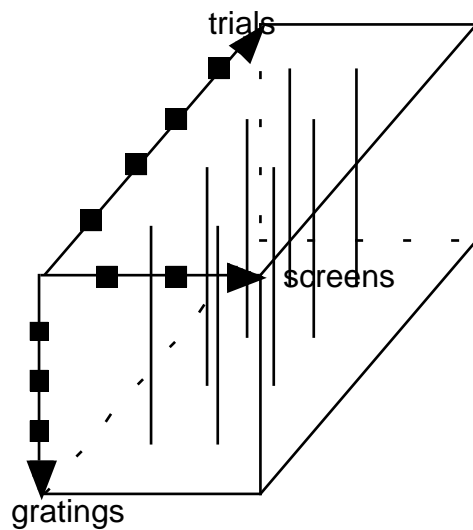


**Figure A-2.**



**Figure A-3.**

What is more useful, though, is to make the contrast vary between trials and between screens. This is shown in Figure 4, where the areas to which each symbol applies are shown as lines. Each symbol applies to one combination of screen and trial in this case. It is also possible to restrict a symbol to a combination of screen, trial, and grating. This is not pictured.



**Figure A-4.**

In reality, the situation is a little more complex. As well as trials, screens, and gratings, there are fusion bars, fixation stimuli and pictures. Your symbols actually apply to a 6-dimensional space.

The indices on the axes for gratings, fusion stimuli, and fixation stimuli are arbitrary. There is no requirement that you use adjacent ones, only that they are in the range 0 to 1000.

To give a symbol a value, you enter something of the form *symbolname=value* in either the trial file or the parameter file. The precise formats of these files are described later. An optional *axis restriction* may follow the symbol name. For example,

- `contrast=0.5`  
sets the contrast for everything to 0.5.
- `contrast(scr=0)=0.8`  
sets the contrast for everything on the left screen to 0.8.
- `contrast(scr=0, stim=2)=0.2`  
sets the contrast of grating 2 on the left screen to 0.2.

(The full list of axis codes is bellow.)

### Controlling what gets displayed

The way the program decides what to do with the symbols you give it is a little complicated. ODE finds the number of trials in an experiment by looking at the trial file (you have one line per trial, this is described later). There are always two screens. The difficult bit is for it to decide how many gratings, fusion bars and fixation stimuli are meant to be on each screen. (You might want three stimuli on one screen and only two on the other.)

The easiest way to explain this is with an example. Assume that ODE needs to find the IDs for

---

the gratings to show on the left screen (screen 0, see the section *The Axes*) of the third trial. These IDs are the values which are given in an axis restriction with a symbol. ODE does the following to find this set:

- find all of the symbol values which apply to all or part of the left screen of the third trial. To understand what the phrase “all or part of” means, think about the figures above. A symbol value may apply to grating 4 of each trial on the left screen. This would intersect with the region of values that the program is looking in, so it would be included. Another value might apply to the grating 2 of each screen for each trial. This applies to the entire region that ODE is looking at, so it is included too.
- ODE would then consider each grating ID mentioned in the set of symbol values that it found to be IDs which are meant to be shown (2 and 4, in the example).

### **How values get chosen**

When ODE wants to find a value for a symbol of a part of the trial it follows the following rules:

- It finds all symbol values in the experiment that are for the desired symbol and match only the desired item. If any of these are found, the last one that was specified (trial file overrides params, params later in the file override earlier params) is used.
- Failing that, more general matches are used. A value that is specified for  $n$  of the desired axis values will be preferred over one that matches  $n-1$  axis values. When there are two or more partial matches on the same number of axis values, the preferred match is undefined.

For example, if ODE is trying to find the value for the diameter of grating 2 on the right screen of the third trial, the program tries to find a value in the trial file for `diam(scr=1, stim=2)`. If this is in the trial file, it will match on three axes (the third match is on the trial number). Otherwise, the program looks in the trial and parameter files for matches on two indexes, then one, then zero. A symbol value that has zero matches would be one such as `diam=5` (found in the parameter file) that applies to the whole experiment

---

## Appendix B: Change list

2.21 KKH Updated ODE: September 2002:

Made a change in P4ButtonInputTranscriber.cpp so that the final record of the keypress record is now added correctly (before I was adding two final records to fix this).

2.21 KKH Updated ODE: August 2002:

In a probe trial, ODE no longer pauses for the IRI after the last repeat in a trial.

The probe trials are timed so that we can use the duration of the trial in the key press record. The timing now starts after the frames have been drawn (without the probe) before the first repeat. There is a discrepancy between the last state change time, and the duration of the trial. I think this is due to the final drawing of the screens with out the probe, and the final sounds ODE makes after the last repeat.

ODE now adds the correct final record to the key press record. ODE sees what the final state change was (either pressing a key or releasing a key) and uses this to determine the state of the keys at the end of the trial. This does not apply to probe trials at the moment because each probe repeat will not end until all the keys have been released.

2.21 KKH Updated ODE: July 2002:

Altered the default sounds so that they are all set to none if no sounds are specified.

Added a new symbol called probeprob. This symbol requires a float that represents the probability of a probe being presented for each trial.

Added error checking for probeprob. The symbol is checked only if the function type is random.

2.21 KKH Updated ODE: June 2002:

I added some error checking to the UTrailChecker.cpp file. ODE now checks to see if the requested experiment is a probe trial. If so, ODE then checks to see whether the probemisstime symbol is present. If the symbol is missing, ODE returns an error informing the observer that this required symbol is missing.

I have implemented new sound features in ODE for probe trials. Two new symbols are used to define the type of beep the computer makes. These are called

---

probesoundduration and probesoundfrequency. I have added error checking to produced an error message if either of these symbols is not defined in a probe trial.

I have added five new symbols, each describing different sound events made by ODE. These are: probestartsound, probehitsound, probemissound, probecrsound, probefasound. These symbols can contain the values: onebeep, twobeep, threbeep or none. If any these symbols are not defined, ODE use onebeep by default for this particular sound event.

## 2.21 KKH Updated ODE: June 2002:

I found a bug in the probe trials functions related to the key press record. Not all key presses were showing up in the key press records for probe trials.

An interrupt service routine is enabled and disabled to allow key handling. In the probe trials, the interrupt service routine is enabled and disabled within each repeat. I modified this so that the interrupt service routine is enabled before we begin looping through each repeat, and the interrupt service routine is disabled after we have finished all the trial repeats. This has solved the problem with the key press record. I have tested ODE since these modifications, and the key press record for probe trials now contains all appropriate key presses.

This bug seemed to be related to the bug that produced incorrect data in the results files for a probe trial. The data would indicate incorrect keys as being pressed (i.e., - key 1 instead of key 2). It seems that this bug has been fixed now that we only enable one interrupt service routine in the probe trial.

I have noticed that the Probe Trial was implemented so that it works with any button, instead of the button that is supposed to be used. This is a bug that should be fixed by enforcing which button is being pressed during a probe trial.

In the function that processes the responses I found code that already produces a complete key press transcript with a final record containing the total trial duration and the key state at this time. I doubled the code that adds the final duration record and discovered that this fixed the problem of there being no entry for the end of the trial. This means that there is a problem somewhere in the process of copying the contents of the transcript to the transcript file. In the meantime, this solution seems stable enough, though it is very confusing to programmers.

In the code mentioned above, it adds a total duration time that does not apply to probe trials. This time was previously taken from the value of the trialLength symbol. In the case of probe trials, this is a big mistake, because probe trials should ignore the

trialLength symbol because probe trails have variable trial lengths, not a static value.

To address the problem of total duration time being set to trialLength, I created a global variable called gProbeTrialDuration in the trials.cpp file. During a probe trial, I time the trial and put this value in the gProbeTrialDuration variable. When I go to call the function that processes the responses, I check to see if the trial is a probe trial or not. If it is, I use the global variable gProbeTrialDuration. If not, I use the value associated with the trialLength symbol. I have checked this new feature of ODE, and it seems to work effectively.

In the case of probes, there is a slight discrepancy between the final time and the last key press plus the IRI {Is there an IRI after the last one? Shouldn't be!!}. In theory, the final time (or total duration time) should be the last key release time plus the IRI time. Because we measure the time it takes to do a task with the same tools implementing the task, our results can never be exact. The time in this case seems to be out about half a second (this may also incorporate time-consuming procedures that I am unaware of). It might be a better solution to derive the total trial duration by just taking the final key release time and adding the IRI duration. This would be neat but not onehundred percent correct.

## 2.21 KKH Updated ODE: June 2002:

I have modified ODE so that it now adds a final record to the key press record to record the final state (which will always be all keys up?) when the trial is over. As of yet, the time used in this last entry will be taken from the trialLength value in the params file. This means that this result will be wrong for probe trials. I do not know if key press records will ever be produced with a probe trial. If they are never used in probe trials, then we don't have to worry about the time in the final record. Otherwise, we will need to derive the total time taken in a probe trial and use this time in the final record.

There also seem to be some strange results when using producing a key press record in a probe trial. For example, sometimes false alarms are ignored, and sometimes they aren't. Another problem is that only one key press is recorded for each presentation even if the key press handling has been set to second press. Once again, if key press records aren't supposed to be produced for probe trials, then this is not a problem.

I added more checking to the UTrialChecker.cpp file. The ODE now checks to see the trial kind is a probe trial. If so, the ODE then checks to see if the probemode symbol is present. If this symbol is not present, the ODE returns an error informing the observer that probe mode type (continuous or flash) has not been specified.

## 2.21 KKH Updated ODE: June 2002:

If a False Alarm occurs in a catch trail, the reaction time will be a negative reaction time. This is the time that the False Alarm occurred minus the time that the Probe Stimulus would have been presented. The delay time is -1.

If a False Alarm occurs in a non-catch hazard trail, the reaction time will be a negative reaction time. This is the time that the False Alarm occurred minus the time that the Probe Stimulus would have been presented. The delay time remains the time it would have taken to present the probe stimulus.

ODE now checks to see if the requested experiment is a probe trial with a flash probe mode. If so, ODE then checks to see if the probeflashtime is present. If this symbol is missing, ODE returns an error informing the observer that this required symbol is missing.

Up to this point, ODE has had no comment format for each function. I have gone through every function in the trails.cpp file and added comments to each function with the format illustrated below.

```

/*****
*****
Function Name:      The actual name of the function
Auther/Date:       who made it and when
Purpose:           what the function is intended to do
Arguments:         what are the inputs of the function
Returns:           what are the output of the function
Comments:          any extra commentry about the function
*****
*****/

```

## 2.21 KKH Updated ODE: May 2002:

Added a new symbol to ODE called probemode. This symbol specifies the way the probe stimulus is presented. The two options are continuous and flash. If the probemode symbol is missing, continuous mode is used by default. We may need to add a warning (or even an error) if this symbol is missing.

Added a new symbol to ODE called probeflashtime. This symbol specifies the amount of time to present the probe stimulus if probemode = flash. If the probe mode is flash,

---

but the probeflashtime is missing, a default value of 200 (a fifth of a second) is used for the probefashtime. We may need to add a warning or error for this particular case.

Added code to the probe trials function that implements both type of probe modes. I took into consideration the possibility of an observer reacting so fast that their reaction occurs during the probeflashtime period in a flash mode trial. If this happens, the probe stimulus disappears instantly, and the trial ends with a hit as a result.

## 2.21 KKH Updated ODE: Dec 2001:

ODE now contains code that produces square wave sounds of a specified frequency and duration. This function has been added to the trials.cpp file. This is not been incorporated into the trials yet, but this should not be a difficult task.

ODE now checks to see what kind of key press has been requested in probe trials (which was not the case with the previous version). The user can now specify single or double key pressing using the ProbeKeyHandling symbol.

With single pressing, we begin a trial by pressing a key and holding it down, and we signal a reaction by releasing the key. With second press we press and release the button to begin the trial, and the press again to signal a reaction. If second press has been specified, ODE now waits for the observer to release the first button press before entering the trial.

ODE now can use both double press and press and release key handling in both Hazard function trials and Random function trials.

## 2.21 KKH Updated ODE:Nov 2001

Updated Version check, so that ODE now checks to see if the requested parameters file contains a reference to the current version (2.21). If not, ODE warns the user, and requests whether or not the user wishes to continue. Take a look at UTrialChecker.cpp to see where this is done. For future versions of ODE, this will need to be updated.

Corrected the bug in printing the version of ODE. The version is printed to two decimal places, so the correct version is displayed (2.21). If we begin to use versions that go up to 3 d.p. the program will need to be updated.

The remaining modifications were generally made to the file trials.cpp, focussing

mostly the the function named DoProbeTrial.

The program was running such that the first probe data result would always be a hit with a duration of 1. I changed this so that the correct result is used. This did not seem to effect ODE, but I have a feeling that this bug had been intentional (possibly created to work around an ever larger bug?). The version prior to 2.21 contains the state of the code before I started modifying anything. This might provide some insight to why we seem to have problems with ODE crashing in probe trials.

ODE did not check for false alarms during a random delay period in a probe trial. This is now fixed ~~so that the ODE does check for false alarms.~~

Removed code that changed any non-Hit duration to -1.

When there is a false alarm, ODE now derives a negative reaction time. This is the time the observer made the false alarm minus the time the stimulus would have appeared.

Previously, in a false alarm, the ODE measured its delay time up to the false alarm instead of when the stimulus would have appeared. This has been corrected so that the delay time is the time it would take for the stimulus to appear (even during a false alarm).

The observer can now miss a probe trial (which was not the case with the previous ODE). This is done by using probemisstime. If this key word is missing from a params files, the ODE uses a default value of 99999999. It is highly possible that a warning should be added to inform the user that the probemisstime is missing. It even possible that an error should occur if the probemisstime is missing.

The next repeat within a probe trial will not commence until the observer has released the button. This has been added in case an observer is holding down a key in a trial, and misses the stimulus, and then continues to hold the key without knowing that the next repeat is about to begin.

ODE used to draw the probe stimulus after a false alarm. This is not needed, and can confuse the user. ODE has been modified so that now if a false alarm occurs, the probe stimulus is never presented.

ODE used to draw the probe stimulus during the Inter-repeat Interval. This has been changed so that the stimulus is no longer drawn during this interval.

Fixed a bug that caused the probe stimulus to be seen after a correct rejection during a probe trial. The probe stimulus is never displayed during a correct rejection now.

ODE has now been implemented so that a Correct Rejection in a Catch Trial will have a delay time of -1. The same value is used for the delay time if a False Alarm occurs in a Catch Trial. This value can be used to flag the occurrence of a Catch Trial when looking through the results of a probe trial. This value has been used because when a catch trial occurs, there is now probe stimulus (as we are trying to catch the user out), so there is now value for the probe stimulus delay.

Previously, ODE would calculate the delay in a Hazard trail on the fly. That is to say, the delay derived through the Hazard function is found precisely at the point at which the delay has passed. In order to take into consideration there being a minimum value to the delay time, the code was implemented so that the program paused for the minimum period of time before the hazard delay is found on the fly. This has two problematic consequences. First, ODE is not checking for false alarms while it is pausing for the minimum period. Second, if we stop the hazard delay prematurely (e.g., because we have a false alarm), then we never know what the delay was going to be. This means we need to wait out the rest of the delay just to find out what value it is going to have. In order to fix this, I modified the probe trials so that they derive the value of the delay value using the hazard function in advance. Then we loop through the minimum delay, checking for false alarms. If none occur, we can then enter a new loop while we wait for the delay to pass (which we have already derived through the Hazard Function). If we exit the loop because of a false alarm, we still know the time it would have taken to display the probe stimulus (that is, if it isn't a catch trial).

2.20 ros updated User Guide: 2001/09/07

In particular, I changed the text so that ``stimuli`` and ``stimulus`` reverted to their usual meanings, instead of just referring to gratings and pictures. ``Fusion figures`` and ``fixation points`` also became stimuli.

notes to any following programmer:

the file with most of the work relating to probe experiments is trials.cpp, the function being doprobetrial. the experiment still suffers from the following problems:

1. sounds - i wanted too allow various sound responses - the code is there but doesn't work. also, code for more interesting sounds should be possible (is done in utrocular disc program on sue's machine) but has not been got working.
2. crashing - on the 1st repeat of the first trial, a cr/miss would cause a crash (in writelinetot???) were it not for an ugly workaround. a better solution is necessary.
3. repeats - robert would like to be able to vary within repeats such things as the grating angle. this should be possible but difficult.

- other useful files:

ODEStimulusFrame.cpp - controls drawing of screen to some degree, called by doprobetrial etc to prepare stimuli on screen & to draw them.

TrialAttributes.cpp/h - modify these to add options to the .param/.trials files. possibly also modify UTrialChecker if want to verify values etc.

ODEInputSource - higher level key handling

PStimulusWave - higher level handling of drawing of waves for gratings - i played around with this in implementing concentric circles & spokes, and started work on spirals here which i didn't finish.

P24BitGraphicsAdapter - if you want to try fixing colour calibration, try starting here, but good luck :)

MultiGratingFrame.cpp - handles drawing of individual gratings, but in a wierd way, so i found it difficult to play with in modifying to allow for probe vs normal gratings.

2.20 mcb last updated: 2000/09/04

Pink/Blue flash at startup removed by disabling the random returns from colour functions (perhaps have as further option???).

probecomb option added.

wavecoord option added to allow grating waves to be concentric circles/spokes.

options for varying beep feedback added but fails to work.

handling of miss/cr/hit/fa improved, output fits with what robert wants better now.

2.14 mcb 2000/09/04

- Probe now more fully implemented.
    - can specify probe stimuli.
    - probe results output using standard format with .results.
    - endtrialkey works within probe.
    - can specify a number of repeats for a probe trial.
  - Optional parameter to specify version of ODE that parameter file was designed for.
- 2.13 mcb 2000/06/12
- New trialkind called probe.
  - Removed menubar from flashing before trial.
- 2.12 mch, mcb 1999/11/16
- Option to make a stimulus immune to pseudorivalry added (see prIgnore).
- 2.11 mch, mcb 1999/10/22
- Motion is now possible (see ‘motiontf’ and ‘fps’).
  - You no longer need to specify all of the input keys. Only the specified ones will be recorded (but all of the columns will still be present in the results and key-press record files).
- 2.10 mch 1999/10/02
- Possible problems: The attribute handling has been completely written. I believe that it is working corectly, but there have been problems. Do not assume that problems with experiments are necessarily yours.
- Fixed a crash when loading experiments with many values.
  - Changed the handling of the attributes in the parameter and trial files. Large experiments are now much faster.
  - All defined input keys must new be pressed before a trial will begin.
  - Added the ability to do pseudo rivalry with a different stimuli on each screen. See ‘trialKind’. You will need to add this symbol to all of you experiments.
  - Errors about luminances that are out of range are now warnings. This is because it is no longer feasible for the program to do complete checks.
- 2.9 mch 1999/08/16
- Beefed up the checking a bit. You no longer need gratings on screens with picture stimuli in order to avoid errors.
  - Added support for combining stimuli on the screen in non-additive ways. See ‘combine’.
  - Fusion figures now respond to the positioning controls.
  - Fixed a bug where gratings were sometimes drawn incorrectly (mostly add odd angles). This problem was visually obvious. You will know if you had it.
  - Changed some terminology:
    - “Stimulus” no longer refers specifically to a grating, but to gratings and

pictures.

- The “stim” code has changed to “grat”. ODE will flag uses of “stim” as errors. This should be little trouble to fix—do a find and replace.
- Because “grating” is being used where “stimulus” was used before, “wave” is used instead of “grating” to indicate the luminance profile of the stimulus. If you forget to change this, the mistake will also be detected by ODE.

### 2.8.3 mch 1999/07/27

- Fixed a bug where real rivalry trials (and the initial frame of pseudo rivalry trials) were drawn incorrectly.
- Added a check for the calibration files. The program no longer quits without comment.
- Changed the files created by ODE (results, counter and key-press records) from CodeWarrior files to BBEdit files.
- Fixed a bug where restricting a symbol to a pict in the trials file would cause a crash when starting the experiment.

### 2.8.2 mch 1999/07/22

- Fixed a bug where ODE would crash while generating some pseudo-rivalry trials.
- Fixed the long-standing problem where, if the program quitted abnormally (force quitting, for example) then the menubar was not properly hidden the next time that it was run.
- Fixed the problem where pseudo rivalry trials with displaced stimuli did not draw correctly.
- One remaining problem is this: The checking done before the experiment starts does not take account of pictures or the location of pictures or gratings in determining the validity of luminances. This means that two stimuli will be treated as lying above each, causing ODE to think that the luminances are much higher than they really are. You will need to use a low lumfac or trick the program to get around this.

### 2.8.1 mch 1999/07/05

- Fixed a data-destroying bug introduced in 2.8.0. It caused the statistical information written into the results file for response key 2 to be the information for response key 1 (yielding two identical sets of results). The key-press records produced were still correct. Needless to say, you should not use 2.8.0 (please delete all copies).

### 2.8.0 mch 1999/06/25

- Added support for positioning pictures and gratings on the screen. hCent and vCent now apply to fixation points, gratings and pictures.
- Added support for four response keys. These are called key1 to key4 and replace leftkey and right key. However, the user is only required to press key1 and key2 to start a trial. The column names in the key-press records have changed to be the numbers of

the keys. This is because it seems more consistent with the results and input files which number the keys.

- Fixed a bug where colour trials were checked wrongly.
- You may now control the shape of a blur using blurshape. blurshape=rect gets rectangular blurs and blurshape=oval gets the old radial blurs. In spite the the names, the fields are still square so you will get circular and square blurs.

#### 2.7.7 mch 1999/06/21

- Because the two images in a real-rivalry trial take a while to appear on some machines, the screens are blanked while the images are drawn to the screens and then unblanked.
- When the results are written, if a key was not pressed at all then, instead of writing out an average time of 0, nothing is written, giving a blank field in StatView.

#### 2.7.6 mch 1999/06/02

- The most important change is that a memory leak has been fixed. This could easily leak multiple megabytes of RAM each trial, causing the program to complain about memory shortages and exit.
- key-press record files now contain a summary of the trial at the top. You may still use these as contrast control files without modification.
- The program no longer crashes when the left and right keys are specified in the trial file and not in the parameter file. Not that all other keys should be specified in the parameter file. This change means that you cannot press the left and right keys to end the alignment phase. You must click the mouse or press return.
- The result columns in the results file now start with 'l' and 'r' (instead of 'v' and 'h'), to better pair them with the keys.
- The text displayed before a trial is controllable now. See the description of the symbol 'pretext'.
- If the program detects an error while it is reading a contrast control file then it will report it as such (previously it complained of a lack of memory).

#### 2.7.5 mch 1999/05/26

- Got more aggressive about using random colours for out-of-range luminances.
- Added a check for fewer than two frames in a pseudo rivalry animation (prnumframes). Using fewer would cause a crash at the start of the trial.
- Made the program cope with colons in key-press record prefixes. Previously this would cause a crash or early exit.
- lumfac must now be specified for an entire trial. It is not acceptable to have different lumfacs for each screen. (This caused problems with pseudo rivalry.)
- Improved the checking of trials:
  - the program did not check that you had specified an angle for fixation points. This caused the trial to be aborted.
  - removed a check that each background on a screen (remember that each stimuli

has its own one) was in the range of allowable luminances. This is because multiple gratings may add to produce an allowable background.

- screens that had only one stimulus were not properly checked. This meant that you could ask for a stimulus with a maximum luminance that was far too high and still be allowed to run the experiment. This has been fixed.
- Pseudo rivalry trials are now checked correctly. This means that it is no longer necessary to use a lumfac of 0.5 when doing pseudorivalry.
- Fixed other bugs:
  - text values for symbols were not read correctly if they contained characters besides letters but were not in quotes. This is fixed. As described in the main part of the manual, you only need quotes if the the text contains spaces or tabs.
  - fixed a bug (that should never have been there) where the averages in the results file were not calculated correctly if the corresponding key was never pressed or pressed once and held down until the end of the trial.

#### 2.7.4 mch 1999/05/14

Three versions came out in quick succession fixing bugs and adding partial support for doing rivalry with pictures.

- The fusion bars and fixation points appear before the trial starts. This is to allow the user to look at the correct part of the screen in advance.

#### 2.7.1 mch 1999/05/03

- Fixed bugs:
  - one where ODE complained if you didn't specify a screen resolution for an entire trial, even if it was present for each screen.
  - Another where some symbols were not detected by the program.
  - Yet another, where the background luminance of a pseudorivalry trial was incorrect.
  - And one more, where the program drew very low luminances using random colours, even when they were in range.
- You may now comment out lines in the trial and parameter files by placing a hash (#) at the start of the line. The comment continues until the end of the line.
- Fixation points and fusion bars are drawn before the trial begins.
- The name of the key-press record files has changed.

#### 2.7 mch 1999/03/22

Contrast-control files can be created from key-press record files. See symbol transPref.

Grayscale pseudorivalry can be done. See the section on Pseudorivalry symbols.

- 
- 2.6 mch 1998/12/04
- Fixed the long-standing problem of ODE crashing when the user entered a bad file name. It now prints a warning and quits.
  - You can take screen shots.
  - The keys are customizable.
  - The file format has changed.
  - Fixation points and fusion bars can now be present in variable numbers and kinds.
  - Unknown symbols are now reported (and the program will refuse to run the trials).
  - Blank lines in the parameters file are tolerated.
  - Added hue and saturation control.
  - Radius, blur, blur prop, mean lum, back lum, hue and sat can all vary between wave (in addition to the old contrast, phase, wave, sf and angle).
  - Can control interaction: lumFac=1
  - Added a (non-cumulative) gaussian blur. The code is 'gauss'.
  - Stimuli can have their grating modulated by other gratings.
- 2.5.8 1998/11/22
- Fixed bug where experiments with a space in their name could not be used (only the text that the user entered before the space was read correctly).
- 2.5.7 1998/11/02
- Fixed a problem with high spatial freq sine waves not adding correctly.
  - Fixed bug where errors specified the stimulus incorrectly.
- 2.5.6 1998/7/24
- The height of the fusion bars can now be controlled with the barHeight symbol. This is a temporary feature which we get replaced with a more general fusion control system. As such the symbol name will likely change.
- 2.5.5
- Allows the user to enter real numbers for all luminances.
  - Places no limit (apart from the size of the screen) on the size of stimuli.
- 2.5.3b 1998/5/31
- This version is beta due to the speculation about colouration bugs.
  - Now uses light mouse files directly and uses 24 bit graphics giving 4000+ levels of gray.
  - The luminances of the lines and background of the alignment screens are now controllable with alignLineLum and alignBackLum.

## 2.5.2 1998/4/24

- The program now enforces the rule that the spatial frequency must produce an integer number of pixels per wave.

## 2.5.1 1998/4/22

- Fixed a bug which caused a blur prop of zero to give a blank stimulus.
- Fixed an incorrect error message stating that sine waves and square waves must have an odd number of pixels. It must really be even.

## 2.5

- The big new feature is support for multiple stimuli on each screen. These can be at different frequencies, contrasts, etc.
- Also now does lots of error checking when it loads a set of trials. It checks that you have not forgotten to specify some symbols (such as not providing a radius) and checks the correctness of some values (notably text ones such as the stimulus and blur codes).
- Allows control of the bars down the side of the screen.
- You can end a trial quickly or extend it. See the section *Running the trials*.
- Blurring is now much faster (at least on a power mac), so even large stimuli can be blurred in most ITIs.
- Don't try running this on a 68k machine (the Mac II for example). There are some odd problems.

---

## Appendix C: Sample files

```
# Call this parameter file probel.param

ODEversion = 2.21

trialKind = probe

# These keys are 1,2,3,4
# For this probe experiment, only one key is used for responses.
# ODE does seem to require that two keys be defined, so key2 is
# defined here. Key1 is defined as a variable in the trials file.

# key1 = 18
key2 = 19
# key3 = 20
# key4 = 21

quitkey = 12
endtrialkey = 53
screenshotkey = 8

# This is the h for hold key--necessary for USB keyboards
extendtrialkey = 4

alignLineLum = 90
alignBackLum = 2

# This is the Inter-trial interval
iti = 1

# trialLength is ignored for probe trials
trialLength = 3

# preText = "Here we go!"

viewDist = 1.00
scrRes = 3307
combine = sum
probecomb = paste

# This is used to produce a key press record
transpref = "br"

# The following specifies spatiotemporal aspects of all stimuli

back = 2
blurshape = oval
blur = none
blurprop = 0.0
contrast = .5
lumfac = 1
meanL = 25
wave(grat=0) = sine
angle(grat=0,scr=0) = 0
angle(grat=0,scr=1) = 90
phase = 0
sat = 0
hue = 0
```

---

```
sf = 2
diam = 2
vcent = 0
hcent = 0
motiontf = 0
fps = 30

# What follows specifies the spatial properties of
# the probe stimulus

stimtype(grat=1, scr=0) = probe
wave(grat=1,scr=0) = flat
angle(grat=1,scr=0) = 45

# What follows specifies the fusion stimuli

fusKind = rect
fusLum(fus=0) = 90
fusHThick = 0.5
fusVThick = 0.0
fusHInner(fus=0) = 2.5
fusVInner = 4

# What follows specifies the fixation stimuli

fixkind(fix=0) = cross
fixLum(fix=0) = 90
fixRadius = 0.3
fixThick = 0.1
fixangle = 45
vcent(fix=0) = 0

# What follows specifies the probe timing

probefntype = hazard
probehazardmin = 300
probehazardmax = 7000
probehazardprob = 0.0004
repeats = 3
iri = 1000
probemisstime = 10000

# What follows specifies other probe details

probekeyhandling = secondpress

# This makes the probe appear for 1 second

probemode = flash
probeflashtime = 1000

# This sets up the sound for the probe trials

probesoundduration = 10
probesoundfrequency = 60
probestartsound = twobeep
probefirstsound = onebeep
probefasound = threebeep
probecrsound = onebeep
probemisssound = threebeep
```

---

Call this file `probe1.trials`. This file runs with `probe1.param`.

<code>hCent(fix=0)</code>	<code>angle(scr=0)</code>	<code>angle(scr=1)</code>	<code>key1</code>	<code>pretext</code>
2.0 90 0	47		.	"Press the . key to show the probe"
-2.0 0 90	6		z	"Press the z key to show the probe"
-2.0 90 0	47		.	"Press the . key to show the probe"
2.0 0 90	6		z	"Press the z key to show the probe"
2.0 90 0	6		z	"Press the z key to show the probe"
2.0 0 90	47		.	"Press the . key to show the probe"
-2.0 0 90	47		.	"Press the . key to show the probe"
-2.0 90 0	6		z	"Press the z key to show the probe"

## Appendix D: Programming Methods

by KeeKim Heng

### D.1 Adding Symbols to ODE

Adding symbols to ODE involves three tasks. First, add the symbol name to ODE. Second, associate the symbol name with a particular variable type. Third, implement error checking so that an error or warning can appear if the symbol is not present in an experiment.

#### D.1.1 Add the Symbol Name

Task 1 is implemented in the `TrialAttributes.h` file. This file contains declarations of all symbols used in ODE. The symbols are in the following groups:

Keys	Alignment	Trial
Probe	Pseudo Rivalry	Screen
General Control	Fusion	Fixation
Stimulus	Gratings	Pictures
AM waves	Internal Values	

It is in this file that we introduce the new symbol variable name, its reference name, and its values if they don't conform to some standard variable type. For example, when I added the `probekeyhandling` symbol, I added it to the probe group of symbols in the form of a variable called `symbol_ProbeKeyHandling`. ODE references the symbol by the variable name (in this case `symbol_ProbeKeyHandling`). The user references the symbol by its actual name (in this case `probekeyhandling`). Following this [{what's this?}](#) are two string (`LStr255`) variables, each referring to the possible string values of the `probekeyhandling` symbol. When the value of the symbol conforms to a standard variable type, we do not need to add these string variables.

#### D.1.2 Associate the Symbol with a Variable Type

Task 2 is implemented in the `TrialAttributes.cpp` files. All symbol associations are contained in a method belonging to the `TrialAttributeHolder` object called `ValueTypeForSymbol`. Just as in the `TrialAttributes.h` file, the symbols are clustered into their appropriate groups. This method is applied to every symbol. If a match can be made with a particular symbol variable name (such as `symbol_InterTrialInterval`) the appropriate variable type of its value is returned (such as `valueType_Float`).

### *D.1.3 Add Error Checking for the Symbol*

Task 3 is implemented in the `UTrialChecker.cpp` file. This file first checks that all of the symbols that need to be defined are defined then checks that all the values are acceptable. Checking is done via several methods, each associated to a particular part of ODE. The best way to see how to implement error checking for a new symbol is to see how it is done for existing symbols. Quite often, we only want to check a symbol if another symbol has been defined, or if another symbol contains a particular value. Examples of this are present in the function that does error checking for the probe functions. Take a look to see how this is done. {Not particularly helpful?}