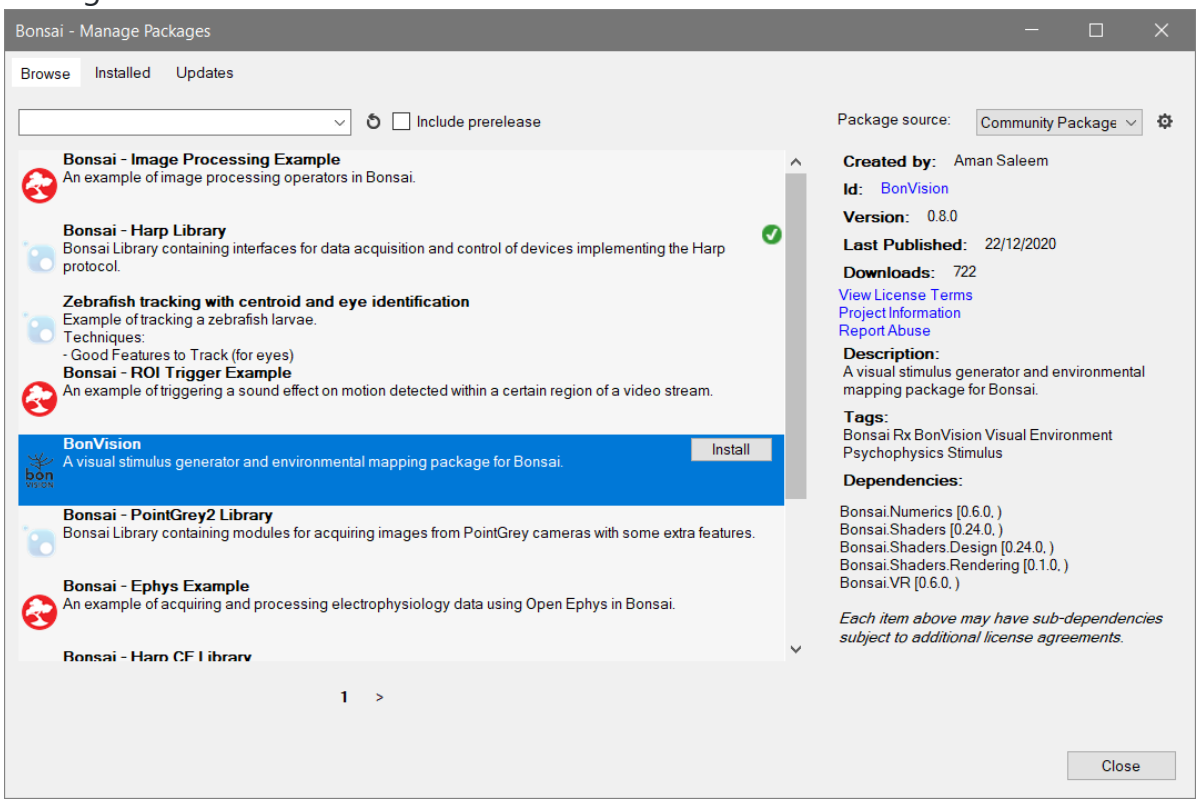
Vision Psychophysics

Getting Started

1. Install the **BonVision** package from the Bonsai Community feed in the package manager.



2. Go through the basic stimuli tutorial at the BonVision website.

Make sure the latest version of the BonVision package is installed for this worksheet

Orientation Discrimination

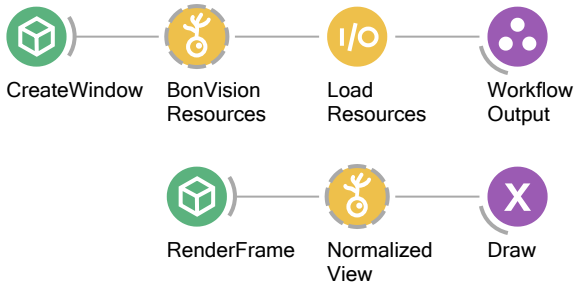
In this worksheet you will build the skeleton of an orientation discrimination vision psychophysics task. In this variant of the task we will present two test gratings in quick succession at different random orientations, and ask the participant to report which of the gratings had the more clockwise orientation. Orientations for each grating will be drawn from a random uniform distribution, and feedback of whether the response was correct or incorrect will be provided visually.

The following set of exercises are to be developed in a single workflow, so do not remove

The following set of exercises are to be developed in a single workflow, so do not remove the elements from the previous exercise from subsequent exercises, unless it is specifically mentioned.

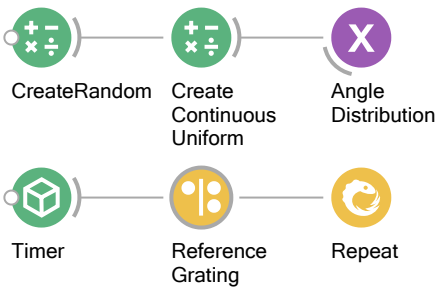
Exercise 1: Random Orientation Grating

To allow sharing screen calibration for all displayed task elements, we start by defining a common BonVision render pipeline.



- Insert a `CreateWindow` source and set the `ClearColor` property to `Gray`.
- Insert the `BonVisionResources` and `LoadResources` operators to preload all built-in BonVision shaders.
- Insert the `WorkflowOutput` operator after `LoadResources` to ensure the workflow terminates when the shader window is closed.
- Insert a `RenderFrame` source. This source will emit a notification when it is time for a new frame to be drawn on the screen.
- Insert a `NormalizedView` operator. This will specify that our stimulus dimensions are resolution independent, aspect ratio corrected, and normalized to the range [-1,1].
- Insert a `PublishSubject` operator and set its `Name` property to `Draw`. We will use these events whenever we need to draw any element on the screen.

The first step in developing our task will be to display a grating in the center of the screen at a random orientation for a specified period of time, and store the value of the orientation, so we can use it later to test the participant.



- Insert a `CreateRandom` source.
- Insert a `CreateContinuousUniform` and set its `Lower` and `Upper` properties to -1 and 1, respectively.
- Insert an `AsyncSubject` and set its name property to `AngleDistribution`.

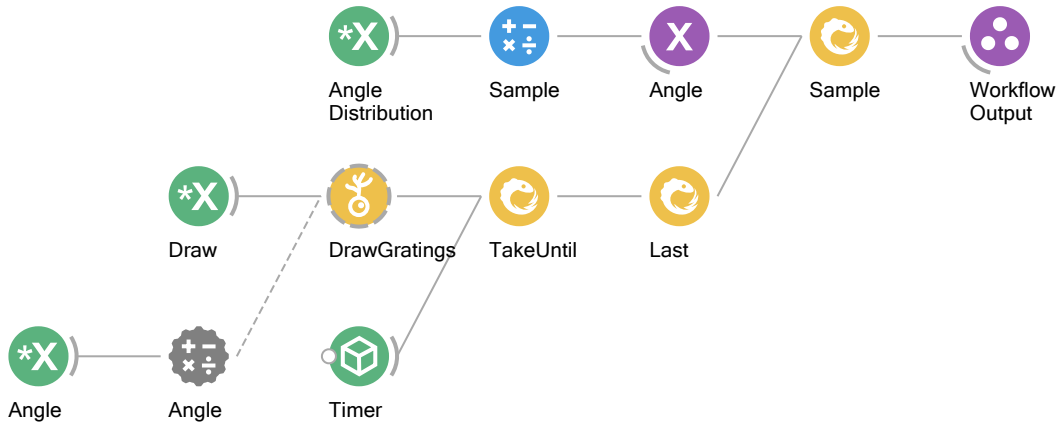
For now, we start by displaying a repeating sequence of random orientation gratings.

- Insert a `Timer (Shaders)` source and set its `DueTime` property to 2 seconds.
- Insert a `SelectMany` operator and set its name to `ReferenceGrating`.
- Insert a `Repeat` operator.

> **Note**: The `Timer (Shaders)` source works exactly like the default `Timer (Reactive)` source, but it counts the time by using the screen refresh time, rather than the operating system time. This can be important for precise timing of screen stimuli, as it avoid clock drift and jitter when synchronizing multiple visual elements, and should be in general preferred when specifying the various intervals used to control elements in the BonVision or Shaders packages.

To implement the `ReferenceGrating` state, we will need to sample a random angle from the angle distribution, use it to initialize the angle property of the gratings, and present the gratings for a specified period of time. At the end, we need to send out as a result the value of the random orientation which was generated.

`ReferenceGrating`:



- Use the `Sample (Numerics)` operator to sample a random orientation value from the `AngleDistribution` subject and store it in a new `AsyncSubject` named `Angle`. This will allow us to reuse the sampled value when drawing the gratings later.
- Subscribe to the `Draw` subject we defined previously and insert a `DrawGratings` operator.
- Externalize the `Angle` property from the `DrawGratings` node and connect the `Angle` subject we created to it.
- Insert a `Timer` operator and set its `DueTime` property to 1 second.
- Insert a `TakeUntil` operator and connect the `DrawGratings` node as the source, and the `Timer` as the trigger.
- Insert a `Last` operator. This will ensure we will get a notification whenever the `Timer` stops the presentation of the stimulus.
- Insert a `Sample` operator following the `Angle` declaration, and connect the `Last` operator as a trigger. This will store the sampled angle value until it is time to return.

- Insert a `WorkflowOutput` operator to specify the final output of the state.

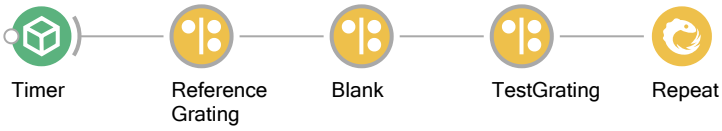
Run the workflow and verify whether the behaviour of the system is correct. Are different orientation values being used for each subsequent presentation of the gratings?

Exercise 2: Reusing stimulus definitions

The second step in defining the contrast discrimination task is to display a second randomly oriented grating in each trial, with a small blank (or masking) period in between. To do this, we want to avoid repeating the entire workflow we designed for our reference grating, so we will make use of the `IncludeWorkflow` operator to reuse our stimulus presentation logic.

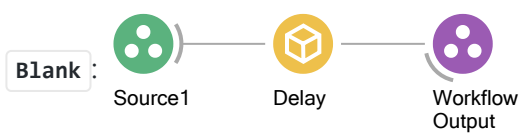
- Inside the `ReferenceGrating` state, select all nodes before `WorkflowOutput`, right-click the selection, and choose the `Save as Workflow` option. Choose `RandomOrientationGrating` as the name for the extension.

After we have our new reusable operator, we can extend the workflow to include the blank period and the second grating stimulus.



- Insert a `SelectMany` operator after the `ReferenceGrating` state and set its `Name` property to `Blank`.
- Insert another `SelectMany` operator after `Blank` with the name `TestGrating`.
- Insert a `Repeat` operator.

For the `Blank` state we will use a simple gap interval where nothing is drawn on the screen. We can do this easily by delaying the transmission of the result of the previous state, before we move on to the next state.

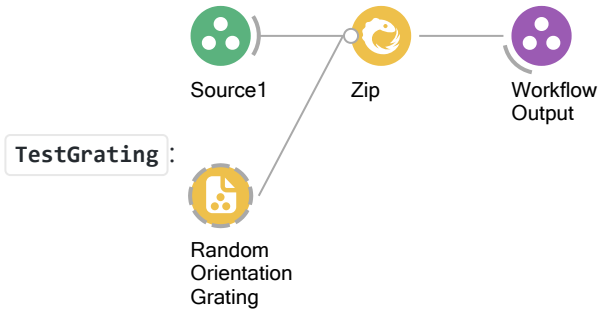


- Insert a `Delay (Shaders)` operator between the input and the output of the state workflow

workflow.

Note: Similar to `Timer (Shaders)`, the `Delay (Shaders)` operator works exactly like the `Delay (Reactive)` operator, but using the screen refresh clock instead of the operating system clock. This also ensures that any delayed notifications are resynchronized with the render loop, in case they were emitted from other external devices.

To implement the `TestGrating` state, we want to reuse our previous `RandomOrientationGrating` extension workflow and simply combine the random generated angle with the angle from the reference grating.

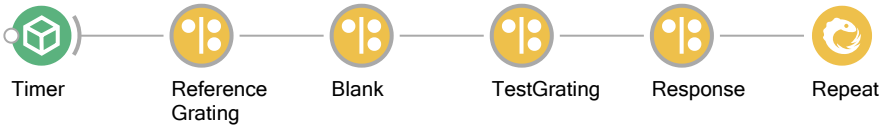


- Insert a new `RandomOrientationGrating` operator from the toolbox and combine it with the input by using the `Zip` combinator. This will generate a pair where the first value is the random angle from the first reference grating, and the second value is the random angle for this test grating.

Run the workflow and validate the random angle pairs are distinct and valid from trial to trial.

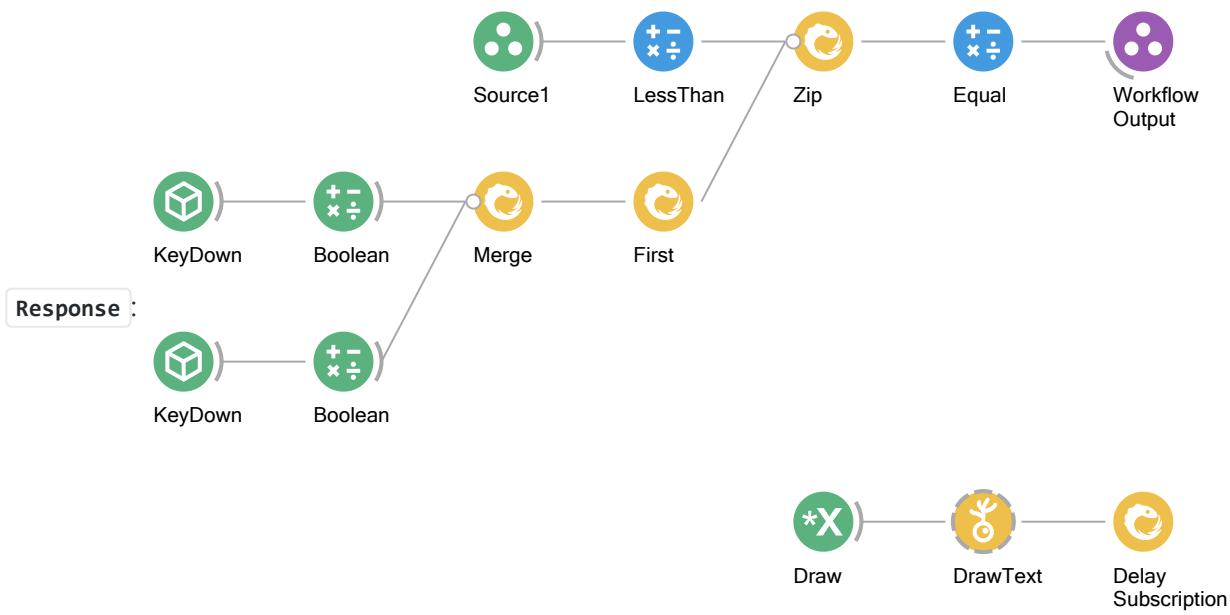
Exercise 3: Collect test response and compute trial outcome

Now that we have our two randomly generated gratings, we need to gather the response from the participant and compare it with the actual situation to determine whether the trial was successful.



- Insert a new `Response` state after the `TestGrating` state using the `SelectMany` operator.

To implement the response gathering state we will use key presses from the participant. We will use the left and right arrow keys to indicate which stimulus had the most clockwise orientation and compare the response with whether or not the first stimulus was more clockwise than the second stimulus.



- Connect the `Draw` subject from the toolbox to a new `DrawText` operator and set its `Text` property to a suggestive question (e.g. `A or B?`). Also edit the `Font` property and make sure the size is at least 72pt for readability.
- Insert a `DelaySubscription (Shaders)` operator and set its `DueTime` property to 1 second.

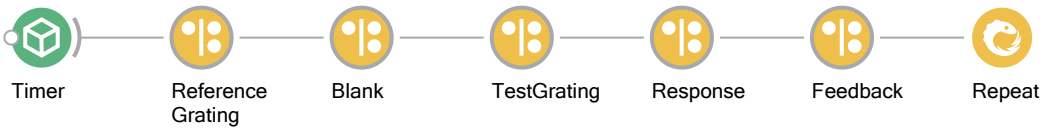
Note: As before, the difference with `DelaySubscription (Reactive)` is that `DelaySubscription (Shaders)` will use the screen refresh time and make sure that all effects of subscription are synchronized with the render loop.

- Insert a `LessThan` operator after the input source node. This will compare the value of the randomly sampled angles for the first and second gratings, respectively, and will return true if the first grating is more clockwise than the second grating (i.e. its angle in radians is smaller than the second grating).
- Insert a `KeyDown (Shaders)` source and set its `Key` property to `Left`.
- Insert a `KeyDown (Shaders)` source and set its `Key` property to `Right`.
- Insert a `Boolean` operator after each of the key press sources and set the `Value` property to `True` for the operator following the left key press.
- Combine the results of both key presses with a `Merge` operator.
- Insert a `First` operator since we are only interested in the first response from the participant.
- Combine the comparison from `LessThan` with the response from the participant using the `Zip` combinator.
- Insert an `Equal` operator to check whether or not the response matches the true

- Insert an `Equal` operator to check whether or not the response matches the true angle comparison. This will be the result of the `Response` state and after it is reported, all other effects of the state will be determined (i.e. the question display).

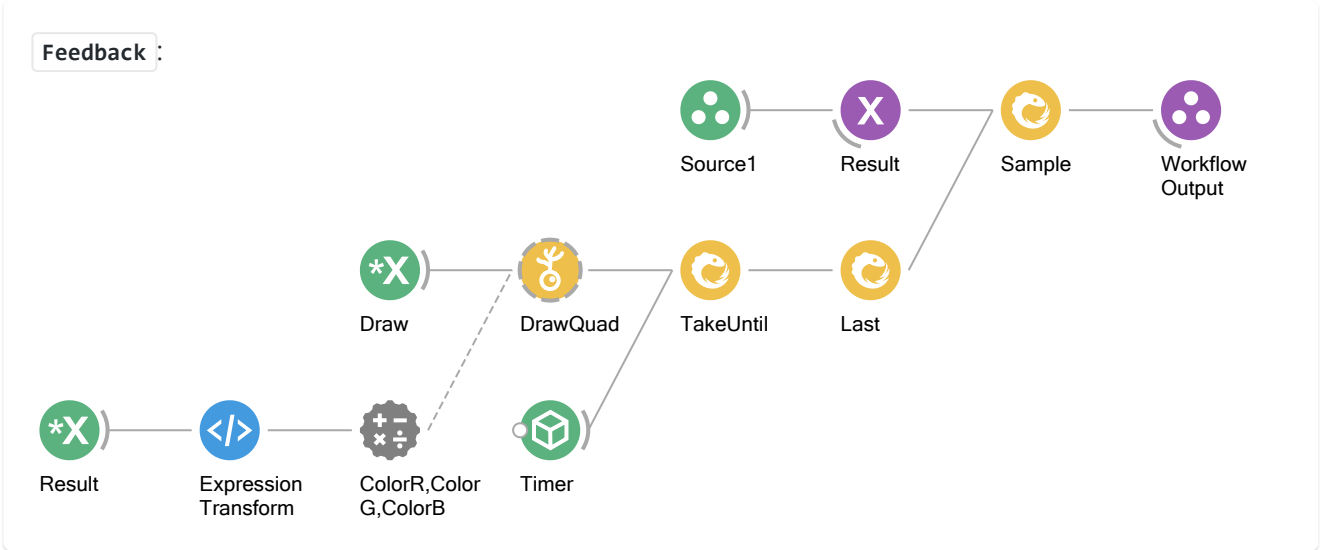
Exercise 4: Present trial outcome feedback to participants

The only step left for finishing our experimental prototype is to report the feedback of each trial back to the participants. We will do this by drawing a colored square, indicating green for a correct response, and red for an incorrect response.



- Insert a new `Feedback` state after the `Response` state using the `SelectMany` operator.

This final state will simply display a quad for a certain period of time, where the color will be modulated by the trial outcome value. We want to store this value until the end of the trial so we can report it for subsequent processing.



- Insert an `AsyncSubject` operator and set its `Name` property to `Result`. This will store the trial outcome result so it can be used to compute the color value of the quad.
- Subscribe to the `Draw` subject and insert a `DrawQuad` operator.
- Externalize the `ColorR`, `ColorG`, and `ColorB` properties from the `DrawQuad` node.
- Subscribe to the `Result` subject and create a new `ExpressionTransform` operator.

In the `Expression` property of the `ExpressionTransform` operator, create a structure holding the RGB color value using the following script:

```
it ? new(0 as R, 1 as G, 0 as B) : new(1 as R, 0 as G, 0 as B)
```

- Connect the `ExpressionTransform` to the externalized properties.
- Insert a `Timer` operator and set its `DueTime` property to 1 second.
- Insert a `TakeUntil` operator and connect the `DrawQuad` node as the source, and the `Timer` as the trigger.
- Insert a `Last` operator. This will ensure we will get a notification whenever the `Timer` stops the feedback presentation.
- Insert a `Sample` operator following the `Result` declaration, and connect the `Last` operator as a trigger. This will store the trial outcome value until it is time to return.
- Connect it to the `WorkflowOutput` operator to specify the final output of the state and trial.

Run the workflow and verify the visual feedback indeed matches the perceived results from each trial.

Exercise 5 (Optional): Measure psychometric data

What is the minimal discrimination threshold for humans in this task? How would you extend the previous workflow in order to assess this?
