Scripting

The Bonsai compiler includes advanced scripting functionality to allow developing custom operators specifically for your project. These scripts will be compiled together with your workflow. The Bonsai editor makes it easier to bootstrap the required infrastructure and the scripts themselves.

This worksheet will walk you through the basics of getting C# scripting to work with Bonsai and a short exercise in building a project with custom scripts.

Getting Started

Install the following:

1. Visual Studio Code.
2. .NET Core SDK.
3. C# extension from the VS Code extensions menu.
4. .NET Framework 4.7.2 Developer Pack.

You might need to reboot your machine after installing these components. {: .notice--info}

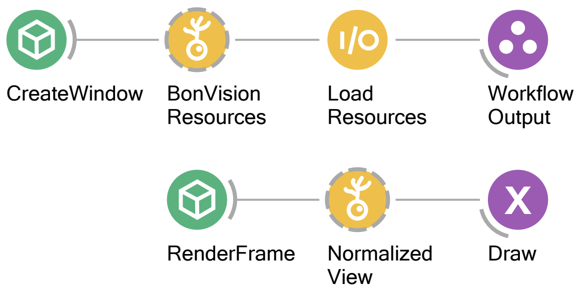
Random Dot Kinematogram

In this exercise, we will implement a random-dot kinematogram (RDK) stimulus using a custom C# script. This visual stimulus is composed of a dense pattern of random dots which are displaced coherently along a primary direction of motion from one frame to the next.

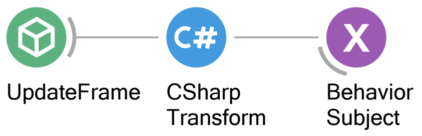
The following set of exercises are to be developed in a single workflow, so do not remove the elements from the previous exercise from subsequent exercises, unless it is specifically mentioned.

Exercise 1: Dot Field

Implement the common BonVision render pipeline shown below. The `Draw` operator should be implemented as a `PublishSubject`.



Next we will create the workflow that will initialize and update the state of the RDK across frames. To do this, we will use a custom operator implemented in C#.



- Set the `Name` property of the `BehaviorSubject` to `DotField`.
- Double-click the `CSharpTransform` operator and follow the instructions to generate a new script file. When prompted, name the script `RandomDotKinematogram`.
- When inside the Visual Studio Code project, look for a pop-up in the bottom-right corner asking about "Reload Extensions". Click the button as soon as it shows up. If you miss the chance you can also click on the small bell on the bottom-right corner of the VS Code window (in the status bar). This will load all necessary dependencies for the script into Visual Studio Code so it can assist you in writing the C# script.

The output type for our script observable sequence should be a `Vector2[]` representing an array of all the positions of the dots to draw in each frame. To get started, you can copy the below infrastructure into the script you have created to initialize a random dot field.

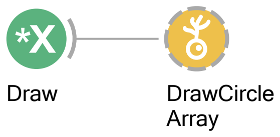
```
using Bonsai;
using System;
using System.ComponentModel;
using System.Linq;
using System.Reactive.Linq;
using Bonsai.Shaders;
using OpenTK;

[Combinator]
[Description("Creates and updates the state of a random dot kinematogram.")]
[WorkflowElementCategory(ElementCategory.Transform)]
public class RandomDotKinematogram
{
    [Description("The number of dots in the random dot kinematogram.")]
    public int DotCount { get; set; }

    public IObservable<Vector2[]> Process(IObservable<FrameEvent> source)
    {
        return Observable.Defer(() =>
        {
            var random = new Random();
            Vector2[] previous = null;
            return source.Select(value =>
            {
                var current = new Vector2[DotCount];
                for (int i = 0; i < current.Length; i++)
                {
                    do
                    {
                        // take points uniformly distributed in the unit circle
                        current[i].X = (float)random.NextDouble() * 2 - 1;
                        current[i].Y = (float)random.NextDouble() * 2 - 1;
                    } while (current[i].LengthSquared > 1);
                }
                previous = current;
                return current;
            });
        });
    }
}
```

This small script simply generates a field of random dots uniformly distributed inside the unit circle every frame. After the script is saved in Visual Studio Code, you can go back to Bonsai and select the menu option `Tools` > `Reload Extensions` to recompile the scripts for your workflow. You will have to do this step every time you change something about your script that you would like to test in the Bonsai workflow.

To visualize the dot field we are generating, we can use the `DrawCircleArray` operator from BonVision.



Make sure to set the `PositionData` property to match the name of the `BehaviorSubject` we defined in the previous step (i.e. `DotField`).

Exercise 2: Kinematogram Parameters

To initialize and update this field as a random dot kinematogram, we will use the following parameters:

- `DotCount`: The number of dots in the random dot kinematogram.
- `Direction`: The direction of movement for coherent dots.
- `Coherence`: The proportion of dots which move together in the coherent direction.
- `DotLifetime`: The number of frames that elapse before a dot disappears and reappears.
- `Speed`: The speed of each dot.

New parameters can be added to the script as properties following the same syntax we used for `DotCount` in the previous exercise. It is possible to add various converters and editors to help us manipulate the property values in the Bonsai environment. To implement the above parameters, for example, you can copy the following declarations into the top of your script (note that `DotCount` was already defined in the previous exercise):

```
    [Description("The number of dots in the random dot kinematogram.")]
    public int DotCount { get; set; }

    [Range(-Math.PI, Math.PI)]
    [TypeConverter("BonVision.DegreeConverter, BonVision")]
    [Editor(DesignTypes.SliderEditor, DesignTypes.UITypeEditor)]

    [Description("The direction of movement for coherent dots.")]
    public float Direction { get; set; }

    [Range(0, 1)]
    [Editor(DesignTypes.SliderEditor, DesignTypes.UITypeEditor)]
    [Description("The proportion of dots which move together in the coherent directio
    public float Coherence { get; set; }

    [Description("The number of frames that elapse before a dot disappears and reappea
    public int DotLifetime { get; set; }

    [Description("The speed of each dot.")]
    public float Speed { get; set; }
```

After inserting these properties and saving the script, select again the menu option `Tools` > `Reload Extensions`. Observe the behaviour of each of the properties in the `RandomDotKinematogram` operator, and try to map it to the property declarations in the script file. You can also delete or change some of the declarations to see the effect in the operator after a reload.

Exercise 3: Dot Lifetime

Our first step towards building a full random dot kinematogram will be to manage the lifetime of each dot. Basically, instead of regenerating the position of each dot in every frame, we want to keep each dot around in the same position for a configurable number of frames (as specified in the `DotLifetime` parameter).

To do this, we can use the `previous` variable which is storing the position of each dot in the previous frame. We also need a new array variable, which we will call `active`, that will store for each dot how many frames of life it still has before it should disappear and reappear somewhere else.

When a dot is regenerated, the `active` value for that dot should be set to `DotLifetime`. Then, for each frame where the dot is active (i.e. the `active` value is larger than zero), keep the same position of the previous frame, and decrease by one the number of frames left for the dot to be regenerated.

▶ Click to show the solution

Exercise 4: Dot Motion

Next, we need to move our dots according to the coherent direction of motion. To do this, we need to add to the previous position of each dot a displacement that will bring it to the new position. Since we later want to generate a distribution of random displacements for some dots, the easiest way is to create a new array variable `velocity` that will keep the vector by which each dot should move.

Note that the `velocity` vector will be a 2D vector in (X,Y) cartesian coordinates that gets added to the dot position. However, the parameters of the kinematogram specify the coherent motion vector using the `Direction` of motion property, specified as an angle, and the `Speed` property specifying the magnitude of the motion vector.

We also need to take into account that points very close to the edge of the random dot field might go outside the field when we add the displacement. In case the dot goes outside of the field, we should make it immediately regenerate.

▶ Click to show the solution

Exercise 5: Dot Coherence

Finally, we want to make sure only a specified proportion of the dots are moving coherently. The remaining points should move in a random direction which is decided when the point is regenerated. The probability of each point to move in the coherent direction is given by the `Coherence` property.

▶ Click to show the solution